UNIVERSIDADE DO MINDELO
DEPARTAMENTO DE CIÊNCIAS ECONÓMICAS E EMPRESARIAIS

CURSO DE LICENCIATURA EM INFORMÁTICA DE GESTÃO

RELATÓRIO DE PROJETO DE LICENCIATURA

Ano lectivo 2013/2014 – 4º Ano

Autor: **Ivanir Dos Santos Gomes, N.º 2277**

Mindelo, 2014

Ivanir Dos Santos Gomes

SISTEMA DE ALERTA POR SMS E EMAIL

(UMCont@ct)

Trabalho apresentado à Universidade do Mindelo como parte dos requisitos para a obtenção do grau de Licenciado em Informática de Gestão.

Orientador:
Professor Doutor João Dias

Mindelo, Dezembro de 2014

DEDICATÓRIA

Dedico este trabalho a todos que de uma forma ou de outro me ajudaram durante estes anos de formação, em especial a minha família, os meus professores, os meus colegas e amigos.

AGRADECIMENTOS

Agradeço aos meus pais pelo carinho, esforço, dedicação e ajuda que sempre me deram ao longo de toda a minha vida.

Agradeço do fundo do meu coração aos meus irmãos, em especial a minha irmã Irany Gomes, pelas ajudas preciosas.

Às minhas tias Júlia Andrade e Natália Nascimento pelas ajudas e hospitalidade prestados.

À minha madrinha pelas carinhosas ajuda e por sempre acreditar em min.

Ao meu professor Doutor João Dias pelas ajudas, ideias, orientação e disponibilidade de recursos para prosseguir com o projeto.

Ao co-orientador Flávio Fortes pela ajuda prestada, dedicação demostrada, e sobretudo pelas ideias que serviram para "revolucionar" o meu projeto.

Ao Manuel Vieira e Filipe Mandl pelas ajudas prestadas durante a fase de desenvolvimento do projeto.

Aos meus colegas de curso pela amizade e diferentes momentos de convívio. Em especial ao amigo e colega Jorge Michael pelas ajudas na elaboração deste relatório.

Aos meus professores, em especial aqueles cujas disciplinas tiveram ação direta no desenvolvimento deste projeto.

RESUMO

A área de trabalho escolhida para o desenvolvimento foi a programação de um sistema de Contacto designada UMCont@ct, integrando as tecnologias de SMS e Email com programação em Java, PHP e base de dados MySQL.

O Sistema será projetado e desenvolvido de uma forma genérica, de forma a permitir difundir informações diversas, de diversas sistemas e empresas quer a nível local, nacional ou mesmo internacional. Para tal o serviço disponibilizará dois URLs GET e POST e um Web Service como forma de comunicação entre os diversos sistemas e a UMCont@ct.

O cérebro do sistema será o servidor Java responsável pela validação, preparação e envio de mensagens de alertas do sistema, com base no protocolo SMTP para envio de Email e protocolo HTTP para a comunicação com o gateway SMS a utilizar.

Pretende-se ainda criar uma aplicação Web em linguagem PHP para administração e configuração de todo o sistema, bem como fazer a gestão de utilizadores do serviço.

Já num vertente cliente o sistema disponibilizará um serviço Web, integrado como um módulo no projeto SIGA (Sistema Integrado de Gestão Académico) da nossa Universidade, que permitirá aos estudantes, docentes e funcionários da instituição agendar notificações SMS e Email sobre diversos assuntos referentes à Universidade do Mindelo, destacando entre eles o serviço de notificação das notas aos estudantes por meio de SMS.

Palavra-Chave

APLICAÇÃO WEB, JAVA, PHP, SMS, EMAIL, CONTACTO, YII FRAMEWORK

ABSTRACT

The work area selected for development was programming contact systems designated UMCont@ct, integrating the technologies of SMS and Email with programming in Java, PHP and MySQL database.

The system will be projected and developed in a generic form, to allow disseminate various information from various systems and companies at the local, national or even international. For this service provide two GET and POST URLs and a Web Service, as way communication between the various systems and the UMCont@ct.

The brain of the system will be Java responsible for the validation, preparations and sending alert messages from the system, based on the SMTP protocol for sending mail and HTTP protocol for communication with the SMS gateway to use.

It is intended yet a creation of an application in PHP language for administration and configuration for the entire system, as well as the service to the users' management.

Already a customer present the system provides a Web service, integrated as a module in SIGA project (Integrated Academic Management) of our University, which will allow students, teachers and employees of the institution schedule SMS and email notifications about a lot issues related to university showing essentially the notification service of grades to students through SMS.

KEYWORDS

WEB APPLICATION, JAVA, PHP, SMS, EMAIL, CONTACT, YII FRAMEWORK

LISTA DE ACRÓNIMOS

GSM – *Global System for Mobile Communication*

HTML – *Hyper Text Markup Language*

HTTP – *HyperText Transfer Protocol*

IMAP - *Internet Message Access Protocol*

JVM – *Java Virtual Machine*

PHP – *Hypertext Preprocessor*

POP – *Post Office Protocol*

SCP – *Secure Copy*

SFTP – *SSH File Transfer Protocol*

SMS – *Short Message Service*

SMTP – *Simple Management Transfer Protocol*

SOAP - *Simple Object Access Protocol*

SQL - *Structured Query Language*

SSH – *Secure Shell*

UML - *Unified Modeling Language*

ÍNDICE

1. INTRODUÇÃO 1

1.1. OBJETIVOS 2

1.2. COMO SURGIU A IDEIA DO PROJETO 3

1.3. MOTIVAÇÃO 3

1.4. METODOLOGIA 4

2. FUNDAMENTAÇÃO TEÓRICA 6

2.1. ENGENHARIA DE SOFTWARE 7

2.1.1. Modelo em Cascata 7

2.2. TECNOLOGIAS UTILIZADAS 9

2.2.1. SMS 10

2.2.2. Email 12

2.3. FERRAMENTAS, AMBIENTE E LINGUAGEM DE PROGRAMAÇÃO 16

2.3.1. Base de Dados MySQL 16

2.3.2. Linguagem Programação Java 18

2.3.3. Linguagem PHP 20

2.3.4. Web Service 20

2.3.5. SoapUI 21

2.3.6. Linguagem Shell Script 22

2.3.7. NetBeans 22

2.3.8. MySQL Workbench 23

2.3.9. Yii Framework 24

2.3.10. WinSCP 24

2.3.11. Putty 25

3. ANÁLISE DO PROJETO A DESENVOLVER 26

3.1. ARQUITETURA DO SISTEMA 27
3.2. MODELAGEM DO SISTEMA 28
3.2.1. Lista de Eventos 29
3.2.2. Diagrama Entidade Relacional 30
3.2.3. Diagramas de Fluxo de Dados 35
3.2.4. Diagrama de Caso de Uso do Sistema 37
3.2.5. Diagrama de Sequência 38
3.3. PROBLEMAS ENCONTRADOS E SOLUÇÕES 41
3.4. TESTES REALIZADOS 44
3.4.1. Testes de Caixa Branca 44
3.4.2. Teste de Caixa Preta 45
3.4.3. Teste de Segurança 45
3.5. RESULTADOS DOS DIFERENTES TESTE 46
4. PROTÓTIPO DESENVOLVIDO 48
4.1. DESCRIÇÃO GERAL DO SISTEMA 49
4.1.1. Autenticação 49
4.1.2. Recuperação de Password 50
4.1.3. Painel de Menu 51
4.1.4. Criação de templates 53
4.1.5. Agendamento de Alertas 54
4.1.6. Importação de Contactos 55
4.1.7. Configuração de gateway SMS 58
4.1.8. Configuração do Servidor SMTP 59
4.1.9. Controlo do Servidor UMC 60
4.1.10. Configuração de Menus 62

4.2. ENVIO DE MENSAGENS 63
4.2.1. Envio de SMS para contactos importados 63
4.2.2. Envio de mensagens por Request http 64
4.2.3. Envio de Mensagem via Web Service 68
4.3. CÓDIGOS DE RESPOSTA DO SISTEMA 72
4.4. LOG DO SERVIDOR DO SISTEMA 73
5. CONCLUSÃO 74
5.1. AVALIAÇÃO DA METODOLOGIA 75
5.2. AVALIAÇÃO DAS FERRAMENTAS UTILIZADAS 75
5.3. CONHECIMENTO ADQUIRIDO 76
5.4. PROPOSTA DE TRABALHO FUTURO 77
6. BIBLIOGRAFIAS 79
7. ANEXO 82
I. CÓDIGO DAS PRINCIPAIS CLASSES DO SERVIDOR JAVA 82
II. MANUAL DO UTILIZADOR UMCONT@CT 107

ÍNDICE DE FIGURAS

Figura 1 - Modelo Cascata 8
Figura 2 - Comunicação cliente Servidor SMTP 13
Figura 3 - Desenho do sistema 28
Figura 4 - Diagrama de classes do sistema 30
Figura 5 - DFD 1º nível 35
Figura 6 - DFD 2º nível 36
Figura 7 - Casos de uso do utilizador administrador 37
Figura 8 - Casos de uso do utilizador cliente 38
Figura 9 - Diagrama de sequência - registo de utilizador 39
Figura 10 - Diagrama de sequência - login no sistema 40
Figura 11 - Diagrama de sequência de envio de alertas 41
Figura 12 - Autenticação no sistema 49
Figura 13 - Recuperação de *password* no sistema 50
Figura 14 - Redefinição de *pasword* 50
Figura 15 - Menus do utilizador visitante 51
Figura 16 - Menus para utilizadores após login 52
Figura 17 - Criação de *template* tipo instantâneo 53
Figura 18 - Criação de template agendado 55
Figura 19 – Visualização de conteúdos de ficheiros do Excel 56
Figura 20 - Importação de contactos do Excel 57
Figura 21 - Atualização de uma chave de importação 58
Figura 22 - Configuração de gateway SMS 59

Figura 23 - Configuração da comunicação SMTP 60

Figura 24 - Formulário de Controlo do servidor Java 61

Figura 25 - Script de execução do servidor Java 61

Figura 26 - Edição de Menus do Sistema 62

Figura 27 - Formulário de envio de SMS para contactos importados 64

Figura 28 - Conteúdo SOAP de retorno de método do Web Service 71

Figura 29 - Utilização do Web service do sistema em PHP 71

Figura 30 - Visualização do log do servidor Java 73

ÍNDICE DE TABELAS

Tabela 1 - Comandos do protocolo SMTP 14

Tabela 2 - Comandos do protocolo POP2 15

Tabela 3 - Comandos protocolo POP3 15

Tabela 4 - Variáveis dos diferentes POST e GET http 65

Tabela 5 - Variáveis do JSON de resposta dos GET e POST 65

Tabela 6 - Métodos disponibilizados pelo Web Service do sistema 70

Tabela 7 - Códigos de resposta do sistema 72

1. INTRODUÇÃO

Qualquer organização que queira destacar-se da concorrência e assim segurar o seu sucesso, terá que investir nas suas relações com os seus clientes. A melhor forma de o fazer, é sem dúvida reforçar a qualidade e eficiência no atendimento, de forma a reforçar o elo cliente empresa e vice-versa.

Atualmente muitas empresas realizam altos investimentos em propaganda, divulgação de produtos e serviços, relacionamento com os clientes, entre outros, porém nem sempre estes investimentos atingem os objetivos, resultados e benefícios almejados. A sociedade está em constante mudança e evolução e para se manter no mercado é necessário estar sempre atualizado e conhecer seu público-alvo utilizando os meios mais eficientes para tal.

Os meios de comunicação e propaganda que sempre foram utilizados podem já não ser eficientes nos dias de hoje. Como exemplo temos a propaganda por *email* ou por panfletos, pois devido à grande utilização das mesmas essas acabaram por se tornar um verdadeiro incomodo ao publico alvo e na grande maioria das vezes é deitado ao lixo, no caso do panfleto.

É nesta óptica que o sistema a desenvolver irá disponibilizar um módulo SMS.

1.1. Objetivos

O objetivo do projeto é o desenvolvimento e implementação de um sistema genérico de contacto por meio de SMS e *Email*. Disponível para vários sistemas ou empresas que queiram utilizar estas tecnologias como meio de comunicação em vários domínios empresariais.

Criar forma de disponibilizar este serviço na Universidade do Mindelo, permitindo aos administradores, funcionários e principalmente aos estudantes um meio de agendar alertas sobre diversos assuntos relevantes à Universidade, principalmente o envio das notas de pautas aos estudantes.

1.2. Como Surgiu a Ideia do Projeto

Esta ideia surgiu depois de uma conversa com o Professor Doutor João Dias, na qual propus a introdução do SMS e *Email* como meio de alerta no projeto de "Digitalização do Livro de Ponto da Universidade", uma vez que já tinha algum conhecimento da utilização dessa tecnologia para envio de notificações, adquirida durante a realização de um estágio na ENACOL (Empresa Nacional de Combustiveis). Como contraproposta o Professor Doutor João Dias propôs-me o desenvolvimento e implementação de um sistema de SMS capaz de notificar os estudantes, docentes e funcionários da Universidade do Mindelo via SMS sobre diversos assuntos relevantes a Universidade do Mindelo.

Já em desenvolvimento, foi-me proposto pelo co-oreintador Flávio Fortes, a remodelação do sistema de forma a torná-lo num sistema genérico de contacto por SMS e *Email*. Sistema Genérico porque iria dessa forma ser não apenas um serviço exclusivo à Universidade do Mindelo, mas sim para todo e qualquer sistema e empresa que necessita de tais serviços. Por isso procedeu-se a transformação de todo o sistema inclusive o nome que passou de SMS UM para UMCont@ct (UMC).

1.3. Motivação

O fator motivador que levou à criação desse projeto, foi primeiramente o desejo de aplicar os conhecimento na área de desenvolvimento, programação e implementação de sistemas, pois é um sector na tecnologia que envolve tanto desafios de negócios quanto técnicos.

Outro fator motivador foi a análise feita na Universidade do Mindelo onde era visível a necessidade de um sistema que permitisse aos estudantes, principalmente das outras ilhas, uma forma de receber as notas dos exames, e outras informações pertinentes.

Além disso, as tecnologias de *Email* e SMS têm-se mostrado cada vez mais, um importante fator de divulgação da informação nos sistemas atuais.

Tinha o desejo de utilizar tecnologias estudadas neste curso de Informática de Gestão, baseadas em programação e modelagem de base de dados, assim como a utilização de novas tecnologias e ferramentas de desenvolvimento de *softwares*.

1.4. Metodologia

Em Engenharia de Software uma metodologia é um conjunto estruturado de práticas que pode ser repetido durante o processo de produção de *software*. Metodologias de Engenharia de Software abrangem muitas disciplinas, incluindo gerenciamento de projetos, e as suas fases como: análise, projeto, codificação, teste, e mesmo a Garantia da Qualidade.

No desenvolvimento deste trabalho optou-se pela engenharia de *software* como metodologias a ser seguida.

Após traçar os objetivos e âmbito do projeto, passou-se por um processo de definição das diferentes fases do projeto, bem como das atividades a serem desenvolvidas em cada uma delas.

De seguida passou-se por uma fase de pesquisa de alguns *softwares* e ou funcionalidades de softwares parecidos com o sistema a desenvolver, como forma de enriquecimento de conhecimentos.

Na fase seguinte fez-se o esboço do sistema de forma a ter uma melhor visão do mesmo e permitir uma melhor avaliação dos requisitos utilizados na fase de desenvolvimento. Esta fase permitiu ter uma melhor visão do sistema e consequentemente alargar as ideias para novas funcionalidades e melhoria das existentes.

De seguida a fase de definição do melhor modo de projeção do funcionamento interno do sistema, baseado em diferentes testes de conexão com os gateways do serviço SMS e da conexão com diversos servidores SMTP para o serviço de *Email*. Testes estes que permitiram ter uma melhor visão dos diferentes protocolos de conexão, e assim optar pelo melhor. Esta fase permitiu determinar de uma melhor forma os requisitos técnicos necessários à codificação do projeto.

Seguiu-se a fase de preparação dos diversos ambientes de desenvolvimento, instalação e configuração dos diversos *softwares* a utilizar na automação dos processos.

Após isso procedeu-se a codificação do protótipo do projeto, acompanhado de diversos testes de caixa branca dos diversos módulos do sistema.

Por fim a fase de teste do protótipo resultante, reajustes necessários ao sistema e finalização do relatório do projeto.

2. FUNDAMENTAÇÃO TEÓRICA

Nesse capítulo iremos discutir as tecnologias utilizadas na implementação do sistema.

2.1. Engenharia de Software

Por se tratar de desenvolvimento de *software* optou-se por escolher a engenharia de Software, como metodologia a ser seguida na implementação do sistema UMCont@ct, pois a engenharia de software consiste na utilização de um conjunto de processos, métodos e técnicas de forma sistemática, disciplinada, organizada e qualificada para se chegar a um produto final de excelente qualidade, uma vez que engloba todo o processo de desenvolvimento, operação e manutenção de um sistema.

Numa abordagem de engenharia de *software*, inicialmente o problema a ser tratado deve ser analisado e decomposto em sub problemas. Para cada sub problemas, uma solução deve ser elaborada. Solucionados os sub problemas isoladamente e por fim integrar as soluções. Para tal, uma arquitetura deve ser estabelecida. Para apoiar a resolução de problemas, procedimentos, métodos e técnicas devem ser utilizados, bem como ferramentas para parcialmente automatizar o trabalho.

2.1.1. Modelo em Cascata

O objetivo de um modelo do processo de desenvolvimento é proporcionar, ao projeto, uma estrutura que reduza os riscos [Miguel, 2002b].

Um projeto sem estrutura é impossível de gerir – não pode ser planeado, não pode ser estimado, não pode ter marcos definidos, o seu progresso não pode ser monitorizado e não pode ser feita qualquer promessa ao cliente, sobre o seu custo, ou a sua qualidade. Um modelo do processo fornece uma estrutura, desenhada com o objetivo de reduzir o risco e a incerteza, e de aumentar a governabilidade do processo de desenvolvimento.

Este modelo de processo, conhecido vulgarmente por modelo em cascata, propõe uma abordagem sistemática, linear e sequencial ao desenvolvimento de *software*. Embora neste

modelo estivesse previsto *feedback loops*, a grande maioria das organizações que o aplica trata-no de forma estritamente linear.

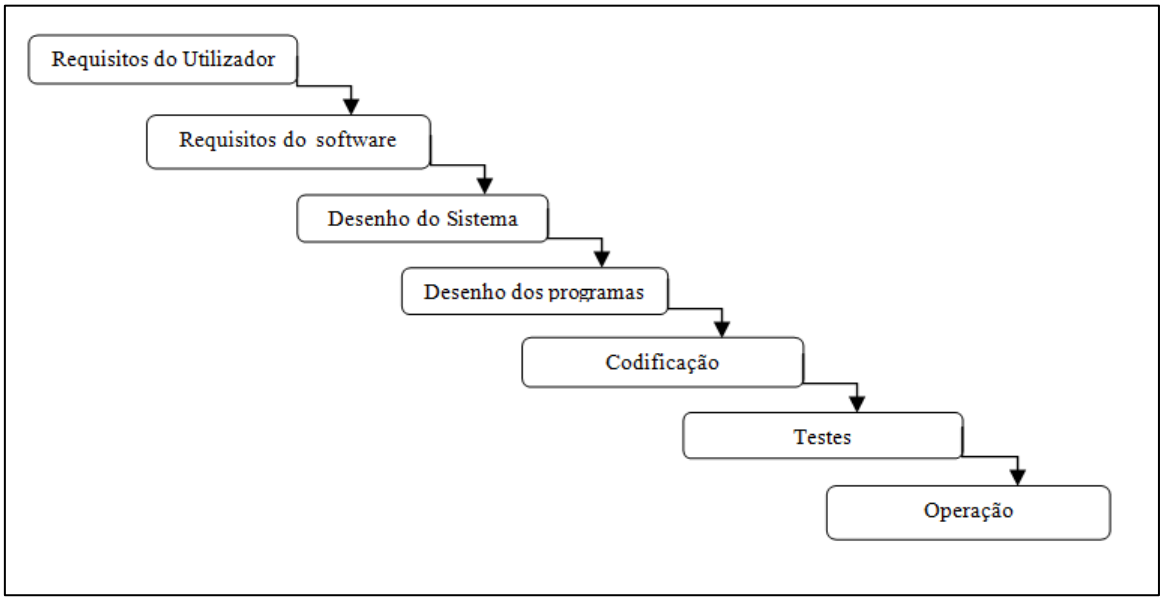

Figura 1 - Modelo Cascata

Requisito do utilizador - Trata-se de uma fase fundamental de engenharia e modelação do sistema de informação empresarial. Consiste no estabelecimento das necessidades e requisitos globais de informação da organização e na subsequente atribuição de um subconjunto ao *software*.

Requisito do *software* - O processo de determinação dos requisitos é, de seguida, intensificado, concentrando-se especificamente no *software* a construir.

Desenho do sistema - Esta fase, denominada igualmente de desenho lógico, visa modelar uma solução de um modo independente da tecnologia, isto é, baseada nas funções do sistema a informatizar e não na tecnologia em que o sistema será implementado. Inclui os modelos dos processos e dos dados.

Desenho dos programas - Esta fase, designada também por desenho físico, visa modelar a solução tecnológica para o problema e é um processo com múltiplos passos, que

se concentra em quatro atributos de um programa de *software* – estrutura dos dados, arquitetura do *software*, interfaces e algoritmos.

Codificação - O desenho físico tem de ser traduzido para uma linguagem própria da máquina.

Testes - Os testes iniciam-se após o código ter sido escrito. Os testes concentram-se na lógica interna do *software* e nas funções externas.

2.2. Tecnologias Utilizadas

A União Internacional de Telecomunicações (UTI) através do relatório anual "*Mesasuring the Information Society*" (MIS), publicada no dia 7 de Outubro de 2013 e que estudou 157 países, estimava que no final do ano 2013 cerca de 2,7 bilhões de pessoas em todo o mundo estariam ligados a internet.

Números divulgados no mesmo estudo mostram a demanda global pelas TICs, produtos e serviços de comunicação e a queda contínua dos preços para os serviços móveis e banda larga, com um crescimento sem precedentes na captação de 3G. Estimava-se que no final de 2013, haveria 6,8 bilhões de assinaturas para telefones celulares – quase o número total de pessoas no planeta.

Analisando as tendências de preços da banda larga em mais de 160 países, o mesmo relatório mostra que em quatro anos – entre 2008 e 2012 – os preços fixos da banda larga caíram em 82%. A maior queda ocorreu nos países em desenvolvimento, onde os preços de banda larga fixa caíram 30% a cada ano entre 2008 e 2011

O mesmo estudo revela que Cabo verde subiu 5 posições no ranking mundial, da 101ª para 96ª, sobre o índice de desenvolvimento das TICs. Em termos percentuais o desenvolvimento atingiu 3.08 % em 2011 e 3.53% em 2012. A nível de Africana

subsariana, o nosso país mantem a 4ª posição atrás de Seychelles, Maurícias e África do Sul.

O estudo apresentado reflete a importância de investimentos nas duas tecnologias de comunicação a utilizar neste sistema (SMS e Email), como forma de garantir uma melhor interatividade entre os diversos *stakeholders* de uma empresa.

2.2.1. SMS

O SMS nasceu na década de 80, mas o primeiro SMS foi enviado a partir de um telemóvel em dezembro de 1992, pela operadora Vodafone, no Reino Unido, com o conteúdo "Feliz Natal".

SMS (*Short Message Service*) que em português significa Serviço de Mensagens Curtas é um serviço rápido e eficiente, utilizado para o envio de mensagens de texto curtos, através de telefones móveis ou pela internet.

Há muito que o SMS deixou de ser popular apenas entre os mais jovens, passando a ser dos meios de comunicação mais utilizados. Já popular em todas as faixas etárias, esta forma de comunicação não escolhe assuntos ou temas, linguagens ou destinatários.

Hoje em dia o SMS é o meio de comunicação mais rápido e eficaz para chegar junto dos nossos destinatários, permitindo a construção de campanhas publicitárias ou simplesmente a transmissão de informação relevante, tudo a preços muito baixos.

Com o aumento exponencial do uso do telemóvel, e consciente de que o SMS tem vindo a ganhar cada vez mais expressão na comunidade empresarial, qualquer investimento neste meio de comunicação é uma mais-valia para qualquer instituição. Acompanhando essa lógica que o projeto pretende disponibilizar este serviço para facilitar a integração de empresas ou serviços com este importante meio de comunicação.

A UMCont@ct fornecerá duas formas de interligaçao com a sua aplicação ou empresa: uma através de um GET e POST HTTP e outro por *Web Service* tornado mais fácil a integração com diversos sistemas, plataformas e linguagens de programação.

O sistema utilizará *templates* de mensagens, que servirão tanto para definir o tipo de comunicação (SMS ou EMAIL), identificar o utilizador de envio, definir o tipo de notificação (*normal* ou *instant*), bem como definir o modelo do corpo da mensagem a hora e o dia de envio em casos de *template* normal.

Para o envio de SMS o sistema irá utilizar os serviços disponibilizados na internet por empresas de *gateway* SMS.

2.2.1.1. Funcionamento do envio de um SMS

Apesar da popularização do serviço, pouca gente sabe o que se passa por trás desta tecnologia. Os telemóveis estão sempre trocando informações com as antenas, tanto para fazer e receber chamadas, quanto para troca de dados. Todas essas informações passam por um canal de controlo, que fornece um "caminho" para que as mensagens sejam transmitidas.

Quando um SMS é enviado ele passa pela torre (ERB - *Estation Radio base*), chega ao Centro de SMS (*SMS Center*) para posteriormente ser encaminhado ao usuário de destino.

O SMSC é responsável por receber e encaminhar mensagens de SME (*Short Message Entity*) para SME, ou armazená-las quando o recipiente não está disponível. O SMSC pode fazer parte da rede GSM (*Global System for Mobile Communications*) ou estar localizado fora dela. Um único SMSC poderia atender serviços de todas as operadoras, contudo, geralmente cada uma costuma manter o seu SMSC, fazendo acordos para que mensagens sejam trocadas entre redes diferentes.

2.2.2. Email

Um correio eletrónico ou, ainda, e-mail e correio-e, é um método que permite compor, enviar e receber mensagens através de sistemas eletrônicos de comunicação. O termo "e-mail" é aplicado tanto aos sistemas que utilizam a Internet e que são baseados nos protocolos POP3, IMAP e SMTP, como àqueles sistemas conhecidos como intranets, que permitem a troca de mensagens dentro de uma empresa ou organização e que são, normalmente, baseados em protocolos proprietários.

O correio eletrónico é considerado o serviço mais utilizado na Internet, e com a massificação do uso da internet a nível mundial, e em Cabo Verde em particular, motivado por fatores como, maior cobertura dos serviços móveis 3G das operadoras de telefonia nacional, redução dos custos deste serviço, criação do projeto *Koneckta*, programa "Internet para todos" e "mundo novo", entre outros, fez com que o uso do correio eletrónico, conhecido como *email*, deixa-se de fazer parte apenas do público empresarial, passando a ser um importante instrumento de comunicação entre os cabo-verdianos, bastando para isso o acesso a internet.

2.2.2.1. Protocolo do Serviço de Email.

O protocolo TCP/IP fornece um conjunto de outros protocolos que permitem uma melhor gerência do tráfico do correio electrónico pela rede. Entre elas destaca-se:

a) **Protocolo SMTP**

SMTP (*Simple Mail Transfer Protocol*) que em português significa Protocolo Simples de Transferência de Correio que é protocolo padrão que assegura à transferência entre o servidor do emissor para o servidor do receptor utilizando conexão ponto a ponto.

Trata-se de um protocolo que funciona em modo conectado, encapsulado numa trama TCP/IP. O correio é entregue diretamente ao servidor de correio do destinatário. O protocolo SMTP funciona graças a comandos textuais enviados ao servidor SMTP (por defeito, para a porta 25). Cada um dos comandos enviados pelo cliente é seguido de uma resposta do servidor SMTP composta de um número e de uma mensagem descritiva.

Abaixo segue um exemplo de uma transação usando o protocolo SMTP entre (C) **contact@uni-mindelo.edu.cv** cliente e o (S) **smtp.uni-mindelo.edu.cv** servidor SMTP.

```
S:  25 mail.uni-mindelo.edu.cv SMTP Ready
C: EHLO uni-mindelo.edu.cv
S: 250 smtp.uni-mindelo.edu.cv
C: MAIL FROM:<contact@uni-mindelo.edu.cv>
S: 250 OK
C: RCPT TO:<gomezivanir@gmail.com>
S:  250 OK
C:  RCPT TO:<gomes_ivanir@domain.com>
S:  550 No such user here
C:  DATA
S:  354 Start mail input; end with <CRLF>.<CRLF>
C:  Subject:  Teste configurações SMTP
C:  Olá,
C:  Esta é a minha mensagem de teste SMTP.C:
C:  Cumptos!
C:  Ivanir Gomes
C:  <CRLF>.<CRLF>
S:  250 OK
C:  QUIT
R:  221 mail.uni-mindelo.edu.cv closing transmission
```

Figura 2 - Comunicação cliente Servidor SMTP

De acordo com as especificações do protocolo SMTP, todos os caracteres a serem enviados são codificados em **ASCII** de 7 bits, e o 8º bit obrigatoriamente zero. Neste sentido para que seja possível o envio de caracteres especiais como os caracteres acentuados e ou anexos, recorre-se aos algoritmos de codificação ou criptografia MIME (*Multipurpose Mail Extension)*, entre elas o **base64** para arquivos de anexo e o ***quoted-printable*** (QP) para caracteres especiais existentes no corpo da mensagem.

Abaixo segue um quadro com o resumo dos principais comandos utilizados no protocolo SMTP, bem como a sua descrição.

Tabela 1 - Comandos do protocolo SMTP

Comando	Descrição
EHLO	Identificação usando o endereço IP ou nome de domínio
MAIL FROM:	Identificação de endereço do remetente
RCPT TO:	Identificação do endereço do destinatário
DATA	Corpo do Email
QUIT	Saia do servidor SMTP
HELP	Lista de comandos SMTP suportado pelo servidor

b) Protocolo POP

O POP *(Post Office Protocol)*, que se pode traduzir por "protocolo de posto dos correios" permite, como o seu nome o indica, recuperar o seu correio num servidor distante (o servidor POP). É necessário para as pessoas não ligadas permanentemente à Internet, para poderem consultar os *email* recebidos offline.

Existem duas versões principais deste protocolo, o POP2 e o POP3, aos quais são atribuídas respectivamente as portas 109 e 110, funcionando com o auxílio de comandos textuais radicalmente diferentes.

Tal como no caso do protocolo SMTP, o protocolo POP (POP2 e POP3) funciona graças a comandos textuais enviados ao servidor POP. Cada um dos comandos enviados pelo cliente é composto por uma palavra-chave, eventualmente acompanhada de um ou vários argumentos, e seguida de uma resposta do servidor POP, composta por um número e por uma mensagem descritiva.

No quadro seguinte é apresentado os comandos utilizados a quando do uso do protocolo POP2.

Tabela 2 - Comandos do protocolo POP2

Comande	Descrição
HELLO	Identificação através do endereço IP do computador remetente
FOLDER	Nome da caixa a consultar
READ	Número da mensagem a ler
RETRIEVE	Número da mensagem a recuperar
SAVE	Número da mensagem a guardar
DELETE	Número da mensagem a eliminar
QUIT	Saída do servidor POP2

O protocolo POP3 gere a autenticação utilizando o nome de utilizador e a palavra-passe, proporcionando assim uma certa insegurança, porque a palavra-passe assim como o *email* circula às claras (de forma não codificada) na rede. De acordo com as especificações do protocolo (RFC1939), é possível codificar a palavra-passe pelo algoritmo MD5, porem sendo este comando opcional, poucos servidores o aplicam.

Este protocolo bloqueia a caixa de correio a quando da sua utilização, impossibilitando assim dois acessos simultâneos a mesma caixa de correio.

Quadro resumo dos principais comandos POP3:

Tabela 3 - Comandos protocolo POP3

Comando	Descrição
USER	Este comando permite a autenticação. Deve ser seguido do nome do utilizador, quer dizer, uma cadeia de caracteres que identificam o utilizador no servidor. O comando USER deve preceder o comando *PASS*.
PASS	O comando PASS permite indicar a palavra-passe do utilizador, cujo nome foi especificado previamente pelo comando USER.
STAT	Informação sobre as mensagens contidas no servidor
RETR	Número da mensagem a recuperar
DELE	Número da mensagem a suprimir
NOOP	Permite deixar a ligação aberta no caso de inatividade
TOP <messageID> <n>	Comando que afixa *n* linhas da mensagem, cujo número é dado em argumento. No caso de resposta positiva do servidor, este devolve os cabeçalhos da mensagem, seguidamente uma linha virgem e por último as *n* primeiras linhas da mensagem.
UIDL [msg]	Pede ao servidor para enviar de novo uma linha que contém informações sobre a mensagem eventualmente dada em argumento. Esta linha contém uma cadeia de caracteres, chamada lista de

	identificadora única, permitindo identificar de maneira única a mensagem no servidor, independentemente da sessão. O argumento opcional é um número que corresponde a uma mensagem existente no servidor POP, quer dizer uma mensagem não apagada)
QUIT	O comando QUIT pede a saída do servidor POP3. Provoca a supressão de todas as mensagens marcadas como apagadas e reenvia o estado desta ação.

c) Protocolo IMAP

O protocolo **IMAP** (*Internet Message Access Protocol*) é um protocolo alternativo ao protocolo POP3 mas que oferece muitas mais possibilidades. Pois permite gerir vários acessos simultâneos, permite gerir várias caixas de correio e ainda permite a triagem de *email* com mais critérios.

2.3. Ferramentas, Ambiente e Linguagem de Programação

Nesse subcapítulo aborda as ferramentas, o ambiente de desenvolvimento e as linguagens de programação utilizados no desenvolvimento do sistema. Para fazer isso fez-se a análise isolado de cada ferramenta ou ambiente. As ferramentas ou ambientes são: a base de dados MySQL; a linguagem de programação JAVA, o *NetBeans*, o *MySQL Workbanch*, a linguagem PHP, o *Yii framework*, o servidor Apache, o *WinSCP* e o *Puty*.

2.3.1. Base de Dados MySQL

MySQL é um sistema de gerenciamento de banco de dados (SGBD), que utiliza a linguagem SQL (Linguagem de Consulta Estruturada, do inglês *Structured Query Language*) como interface. É atualmente um dos bancos de dados mais populares, com mais de 10 milhões de instalações pelo mundo.

Características

- Portabilidade (suporta praticamente qualquer plataforma atual);
- Compatibilidade (existem drivers ODBC, JDBC e .NET e módulos de interface para diversas linguagens de programação, como *Delphi*, Java, C, C++, C#, *Visual Basic*, *Python*, *Perl*, PHP, ASP e *Ruby*);

- Excelente desempenho e estabilidade;
- Pouco exigente quanto a recursos de novos hardware;
- Facilidade no manuseio;
- É um *Software* Livre com base no GPL (entretanto, se o programa que utiliza o *Mysql* não for GPL, uma licença comercial deverá ser adquirida);
- Contempla a utilização de vários *Storage Engines* como *MyISAM*, *InnoDB*, *Falcon*, BDB, *Archive*, *Federated*, CSV, etc;
- Suporta controlo transacional;
- Suporta a *Triggers*;
- Suporta Cursors (Non-Scrollable e Non-Updatable);
- Suporta Stored Procedures e Functions;
- Replicação facilmente configurável;
- Interfaces gráficas (*MySQL Toolkit*) de fácil utilização cedidos pela *MySQL* inc.

A sua escolha é justificada pela facilidade de integração com as linguagens de programação JAVA e PHP, que são as linguagens escolhidas para o projeto, pela sua rapidez, por ser um *software* livre, e mais importante ainda por estar disponível juntamente com o servidor Apache num único pacote, através do *WampServer* para ambientes *Windows* ou no *XampServer* para ambientes Linux, o que permitirá uma administração e configuraçao do sistema centralizada numa única máquina.

2.3.2. Linguagem Programação Java

Java é uma linguagem de programação orientada a objeto desenvolvida na década de 90 por uma equipa de programadores chefiada por James *Gosling*, na empresa *Sun Microsystems*.

Diferentemente das linguagens convencionais, que são compiladas para código nativo, a linguagem Java é compilada para um *bytecode* que por sua vez é executada por uma máquina virtual.

Característica

A linguagem Java foi projetada tendo em vista os seguintes objetivos:

- Orientação a objetos - Baseado no modelo de Simular
- Portabilidade - Independência de plataforma - "escreva uma vez, execute em qualquer lugar" (*"write once, run anywhere"*).
- Recursos de Rede - Possui extensa biblioteca de rotinas que facilitam a cooperação com protocolos TCP/IP, como HTTP e FTP
- Segurança - Pode executar programas via rede com restrições de execução

Além disso, podem-se destacar outras vantagens apresentadas pela linguagem:

- Sintaxe similar a C/C++;
- Facilidades de Internacionalização - Suporta nativamente caracteres Unicode;
- Simplicidade na especificação, tanto da linguagem como do "ambiente" de execução (JVM);
- É distribuída com um vasto conjunto de bibliotecas (ou APIs);
- Possui facilidades para criação de programas distribuídos e multitarefa (múltiplas linhas de execução num mesmo programa);
- Deslocação de memória automática por processo de coletor de lixo;
- Carga Dinâmica de Código - Programas em Java é formada por uma coleção de classes armazenadas independente e que podem ser carregadas no momento de utilização

Esta linguagem foi a escolhida para a implementação do servidor responsável por todo o processo de tratamento e envio das mensagens no sistema. A sua escolha foi baseada nas características da linguagem acima descrita, bem como pelo facto de ser a linguagem predominante do curso de Informática de Gestão na Universidade do Mindelo, e também pelo facto de ser a linguagem que melhor domino, e que me permitirá um maior grau de independência de terceiros durante a codificação do sistema.

2.3.2.1. Concorrencia e Multithreads

Em um ambiente de múltiplas threads (multithread), não é necessário haver vários processos para se implementar aplicações concorrentes. No ambiente multithread, cada processo pode responder a várias solicitações concorrentemente ou mesmo simultaneamente, se houver mais de um processador.

Em um ambiente multithread, não existe a idéia de um programa, mas de threads. O processo, neste ambiente, tem pelo menos um thread de execução, podendo compartilhar o seu espaço de endereçamento com inúmeros threads, que podem ser executados de forma concorrente e/ou simultânea, no caso de múltiplos processadores.

Quando uma thread está sendo executada, o contexto de *hardware* está armazenado nos registradores do processador. No momento em que uma thread perde a utilização do processador, o sistema salva suas informações. Threads passam pelos mesmos estados que um processo, ou seja, execução, espera e pronto.

A grande diferença entre subprocessos e threads é em relação ao espaço de endereçamento. Enquanto subprocessos possuem, cada um, espaços independentes e protegidos, threads compartilham o mesmo espaço de endereçamento do processo, sem nenhuma proteção, permitindo que uma thread possa alterar dados de outra thread. Apesar dessa possibilidade, threads são desenvolvidos para trabalhar de forma cooperativa, voltados para desempenhar uma tarefa em conjunto.

2.3.3. Linguagem PHP

PHP (***H**ypertext **P**reprocessor*) originalmente (***P**ersonal **H**ome **P**age*) é uma linguagem interpretada, livre, usada originalmente apenas para o desenvolvimento de aplicações presentes e atuantes no lado do servidor, capazes de gerar conteúdo dinâmico na *World Wide Web*. Figura entre as primeiras linguagens passíveis de inserção em documentos HTML, dispensando em muitos casos o uso de arquivos externos para eventuais processamentos de dados. O código é interpretado no lado do servidor pelo módulo PHP, que também gera a página *Web* a ser visualizada no lado do cliente. A linguagem evoluiu, passou a oferecer funcionalidades em linha de comando, e além disso, ganhou características adicionais, que possibilitaram usos adicionais do PHP, não relacionados a *Websites*. É possível instalar o PHP na maioria dos sistemas operacionais, gratuitamente.

O que distingue o PHP de algo como *JavaScript* no lado do cliente é que o código é executado no servidor, gerando HTML que é então enviado para o cliente.

A vantagem no uso de PHP está no fato de ser extremamente simples para um iniciante, e oferecee muitos recursos para o programador profissional.

2.3.4. Web Service

Web Service é uma tecnologia baseada em XML e HTTP cuja principal função é disponibilizar serviços interativos na WEB que podem ser acedidos (ou consumidos) por qualquer outra aplicação independente da linguagem ou plataforma em que a aplicação foi construída. O SOAP (*Simple Object Access Protocol*) é o protocolo padrão universal utilizado para a troca de mensagens entre as aplicações consumidoras e o *Web Service*. O *Web Service* expõe as suas operações por meio de um tipo de esquema XML chamado

WSDL (Web *Service Description Language*). No documento WSDL estão definidas todas as interfaces, operações, esquemas de codificação, entre outros, ou seja, um documento WSDL define um esquema XML que descreve um *Web Service*.

Um exemplo de utilização de *Web service* é a implementação de um serviço de pagamentos *online*, em que é preciso uma conexão segura entre a plataforma a desenvolver e o provedor do serviço bancário de pagamento.

2.3.5. SoapUI

SoapUI é uma ferramenta *open source* escrita em Java cuja principal função é consumir e testar *Web Services*. O *SoapUI* facilita todo o processo de criação e depuração dos testes por meio de uma interface gráfica visual e intuitiva.

Dentre as suas principais características, podemos destacar as seguintes:

- Importação e geração automática das requisições descritas no WSDL;
- Capacidade de gerenciar um número ilimitado de requisições para cada operação;
- Gerenciamento de múltiplo *endpoints* para cada Web Service;
- Validação das requisições e respostas contra as suas definições no WSDL;
- Testes funcionais, carga e *stress*;
- Execução de diversos testes em paralelo;
- Editores com *syntax highlight* e formatação automática;
- Suporta expressões XPATH;
- Suporta criação de testes complexos utilizando *scripts Groovy*;

2.3.6. Linguagem Shell Script

Shell script é uma linguagem de *script* usada em vários sistemas operativos (operacionais), com diferentes dialetos, dependendo do interpretador de comandos utilizado. Um exemplo de interpretador de comandos é o *bash*, usado na grande maioria das distribuições GNU/Linux.

A maior parte dos utilizadores classificam *Shell script* como uma linguagem de fácil aprendizagem. O primeiro passo é, saber o que se deseja fazer, então ver qual o código que executa este comando em *shell* e ali criar, basta escrever o código em algum editor de texto e salvar.

Uma vez criado, um *Shell Script* pode ser reutilizado quantas vezes for necessário. Seu uso, portanto, é indicado na automação de tarefas que serão realizadas mais de uma vez. Todo sistema Unix e similares são repletos de scripts em *shell* para a realização das mais diversas atividades administrativas e de manutenção do sistema.

Os arquivos de lote (*batch* - arquivos *.bat) do *Windows* são também exemplos de *Shell Scripts*, já que são escritos em linguagem interpretada e executados por um *Shell* do *Windows*, em geral o command.com ou hoje em dia o cmd.exe. Os *Shells* do Unix, porém, são inúmeras vezes mais poderosos que o interpretador de comandos do *Windows*, podendo executar tarefas muito mais complexas e elaboradas.

2.3.7. NetBeans

É um ambiente absolutamente livre para desenvolvimento de *software* com código-fonte aberto, o que suporta as seguintes linguagens de programação: C, C + +, Java, PHP, *Groovy*, *JavaScript*, etc. O ambiente fornece aos desenvolvedores ferramentas necessárias para criar profissional, empresarial, *desktop*, aplicações móveis e de internet. NetBeans

suporta as seguintes funções: perfis, destaque de sintaxe de cor, autopreenchimento, modelos de código definidos, etc.

Principais características:

- *Open Source*;
- Suporte para as seguintes linguagens de programação: C, C#, C++, Java, PHP, *Groovy*, *JavaScript*, etc.;
- A capacidade de criar vários tipos de aplicações;
- Apoios: *refactoring*, perfis, destaque de sintaxe cor, *auto-completar*, modelos de código definidos etc.

2.3.8. MySQL Workbench

MySQL Workbench é uma ferramenta gráfica para modelagem de dados, integrando criação e *designer* que pretende ser uma evolução do já famoso *DBDesigner4*. A ferramenta possibilita trabalhar diretamente com objetos *schema*, além de fazer a separação do modelo lógico do catálogo de banco de dados.

Toda a criação dos relacionamentos entre as tabelas pode ser baseado em chaves estrangeiras. Outros recursos que a ferramenta possibilita são a realização da engenharia reversa de esquemas do banco de dados, e a gerência de todos os scripts em SQL.

MySQL Workbench apresenta opções que não o deixa em desvantagem quando se compara com outros programas para modelagem de dados. A ferramenta faz sincronia dos modelos de banco de dados, importa e exporta modelos do DBDesigner4, bem como exporta scripts SQL.

A modelagem dos seus bancos de dados pode assumir níveis conceituais, lógicos e físicos. MySQL Workbench apresenta uma arquitetura extensível, sendo possível visualizar a representação de tabelas, funções, entre outros.

2.3.9. Yii Framework

Yii é um *framework* de alta performance em PHP que utiliza componentes para o desenvolvimento de pequenas/médias/grandes aplicações Web. Permite máxima reutilização de códigos na programação Web e pode acelerar significativamente o processo de desenvolvimento. O nome Yii (pronunciado I ou Di) representa as palavras *Yes, It Is*! (Sim, isto é!), em resposta às mais comuns perguntas (Isto é seguro? Isto é rápido? Isto é profissional? Isto é certo para o meu próximo projeto?) sobre essa plataforma.

O DAO (*Data Access Objects*) do *Yii*, fornece uma API genérica para aceder dados em diferentes tipos de bancos de dados. Como resultado, pode-se alterar o sistema de banco de dados sem haver a necessidade de alterar o código que utiliza DAO para fazer o acesso.

O DAO do Yii é feito utilizando a extensão PHP Data *Objects* (PDO), que fornece um acesso de dados unificado para muitos SGBD populares, tais como *MySQL* e *PostgreSQL*. Por esse motivo, para utilizar o DAO no Yii, a extensão PDO deve estar instalada, bem como o driver PDO específico para o banco de dados utilizado (por exemplo, PDO_MYSQL).

O DAO no Yii, consiste, basicamente, das quatro classes seguinte:

- ***CDbConnection***: representa uma conexão com o banco de dados.
- ***CDbCommand***: representa uma instrução SQL a ser executada no banco de dados.
- ***CDbDataReader***: representa um conjunto de registos retornados.
- ***CDbTransaction***: representa uma transação (*transaction*) no banco de dados.

2.3.10. WinSCP

WinSCP (*Windows Secure CoPy*) é um SFTP (*SSH File Transfer Protocol*) livre e de código aberto, SCP e FTP clientes para Microsoft Windows. Sua principal função é a transferência segura de arquivos entre um local e um computador remoto. Além disso, *WinSCP* oferece básico gerenciador de arquivos e funcionalidade de sincronização de arquivos. Para transferências seguras, utiliza Secure *Shel l*(SSH) e suporta o protocolo SCP, além de SFTP. É, amplamente, utilizado para transferência de arquivos entre ambiente Windows e ambientes Linux.

2.3.11. Putty

O PuTTy é um *software* de emulação de terminal, grátis e de código livre. Suporta SSH, destinado a suportar o acesso remoto a servidores via *shell* seguro e a construção de "túneis" cifrados entre servidores.

PuTTY foi originalmente escrito para o Microsoft Windows, mas foi importado para vários outros sistemas operacionais. Versões oficiais só estão disponíveis para algumas plataformas Unix, e em desenvolvimento para o clássico Mac OS e Mac OS X, e as versões não-oficiais foram desenvolvidas para plataformas como *Symbian* OS e *Windows Mobile*.

Para usar o PuTTY não é necessário sua instalação, pois ele roda diretamente pelo arquivo executável Putty.exe.

3. ANÁLISE DO PROJETO A DESENVOLVER

Neste capítulo farei a análise dos diversos diagramas que ajudaram no processo de estruturação do projeto como forma de retratar o fluxo e conteúdo das informações utilizadas pelo sistema, dividir o mesmo em partições funcionais e comportamentais e descrever a essência de aquilo que será desenvolvido.

A análise estruturada é uma atividade de construção de modelos. Utiliza uma notação que é própria ao método de análise estruturada com a finalidade de retratar o fluxo e o conteúdo das informações utilizadas pelo sistema, dividir o sistema em partições funcionais e comportamentais e descrever a essência daquilo que será construído.

3.1. Arquitetura do Sistema

O sistema será composto por um servidor central que fará todo o processo de validação, preparação e envio dos diferentes tipos de alertas do sistema. Após o cadastro do cliente no sistema, será disponibilizado as credenciais de acesso ao envio de alerta no sistema.

Para permitir a conexão entre os diversos sistemas e a UMcontact, será disponibilizado um *Web Service* e GET e POST HTTP aos clientes. O armazenamento será feito em base de dados MySQL onde posteriormente serão tratados pelo servidor do sistema.

Para que seja possível envio de alertas será necessário que o cliente crie *templates* de mensagens. Os *templates* irão conter diversas informações em que se destacam o dia e hora de envio da mensagem, identificação do utilizador, identificação do sistema do cliente, controlo ou não de caracteres em SMS, definir o tipo mensagem (SMS ou Email), definir o tipo de envio (agendado ou instantâneo), entre outros.

A interface Web do sistema permitirá ao cliente consultar histórico e estado de créditos no sistema. Além disso será possível a consulta do *status* de qualquer alerta enviada no sistema, para tal será fornecido ao cliente, no momento da requisição de envio, uma identificação única para cada solicitação de envio. Essa identificação servirá depois para consulta de *status* da alerta junto ao sistema.

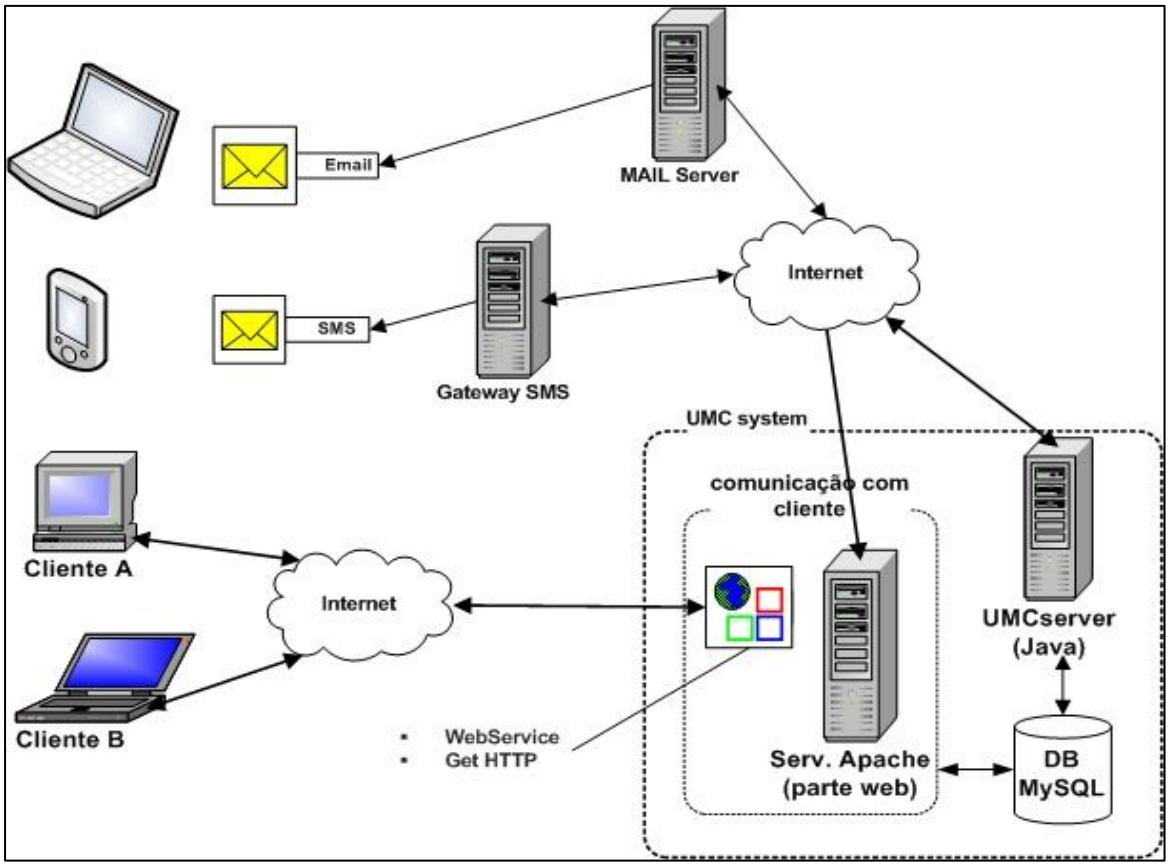

Figura 3 - Desenho do sistema

3.2. Modelagem do Sistema

Para uma melhor compreensão do sistema a desenvolver, farei a listagem dos eventos que podem ocorrer no sistema e descrição tanto do Diagrama de Fluxo de Dados (DFD) geral do sistema, o diagrama de use case do sistema, bem como de diversos diagramas de estado das principais ações do sistema.

3.2.1. Lista de Eventos

A lista de eventos relaciona os factos do ambiente externo que o sistema deve obrigatoriamente reconhecer. Abaixo é listado todos os eventos do Sistema.

- Cadastrar no sistema
- Fazer login no sistema
- Recuperar senha
- Fazer *logoff* no sistema
- Ativar e desativar utilizadores
- Editar dados pessoais
- Alterar senhas
- Administrar servidor do sistema
- Configurar Gateway SMS
- Ativar ou desativar Gateway SMS
- Configurar serviço de *Email*
- Ativar ou desativar serviço de *Email*
- Criar *templates* de mensagens
- Editar *templates* de mensagens
- Enviar alerta no sistema
- Agendar alertas no sistema
- Consultar Histórico de envios
- Importar contactos móveis do Excel
- Enviar alertas SMS de contactos importados
- Agendar alertas SMS de contactos importados

3.2.2. Diagrama Entidade Relacional

A modelagem entidade relacionamento foi desenvolvida por Peter *Chen* e publicada em um artigo de 1976. Entretanto, variantes da ideia existiram anteriormente e, posteriormente, foram imaginadas como entidades de dados de supertipo e subtipo e relacionamentos de uniformização.

O processo é modelado como componentes (entidades) que são ligadas umas às outras por relacionamentos que expressam as dependências e exigências entre elas. Entidades podem ter várias propriedades (atributos) que as caracterizam. Diagramas criados para representar graficamente essas entidades, atributos e relacionamentos são chamados de diagramas entidade relacionamento.

Para este sistema é proposto o seguinte modelo conceitual de dados:

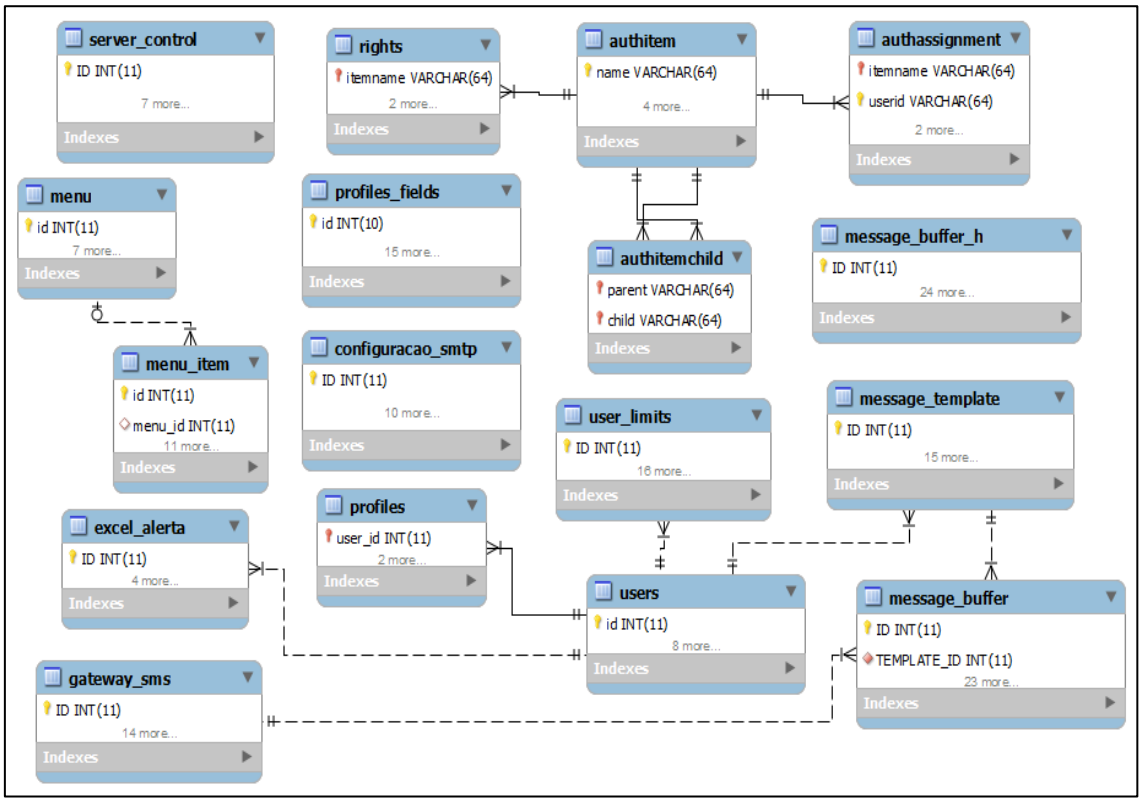

Figura 4 - Diagrama de classes do sistema

O DER fornece o seguinte dicionário de dados:

- **users** – Guardar as informações relativos aos utilizadores do sistema.
  - id – chave primária da tabela.
  - username – nome único que identifica cada utilizador no sistema.
  - password – senha criptográfica do utilizador.
  - email – email de contacto do utilizador.
  - activkey – código de ativação da conta do utilizador.
  - create_at – define a hora e data de criação do utilizador no sistema.
  - lasvisit_at – define a hora e data da ultima login do utilizador no sistema.
  - superuser – define se utilizador em permissão de administrador.
  - status – define o status da conta do utilizador.

- **message_template** – guarda as informações dos *templates* de mensagens de cada utilizador.

  - ID – identificação do *template* (chave primária)
  - DESCRIPTION – permite identificar o *template* através dos seus atributos.
  - STATUS – defina o status do *template* (ativo/inativo).
  - INDICATIVE – defina o indicativo *default* para os números móveis.
  - TEMPLATE_TYPE – define o tipo de *template* (normal/instante)
  - MSG – defina o protótipo do corpo da mensagem.
  - OWNER_SYSTEM – defina o sistema do cliente que utiliza o *template*.
  - OWNER_USER – define o utilizador do *template*.
  - MSG_TYPE – define a tecnologia utilizada no envio (SMS/Email)
  - CONTROL_CARACTER – permite ativar ou desativar o controlo de caracteres utilizados nas mensagens SMS.
  - SEND_DAY – permite definir o (s) dia (s) de envio da mensagem.
  - SEND_HOURS – permite agendar a hora (s) de envio da mensagem.

- **message_buffer** – contem informações das mensagens em processamento ou agendados
  - ID – identificação única do *template* (chave primária)
  - TEMPLATE_ID – identificação do *template* da mensagem
  - GATEWAY_ID – identificação do *gateway* utilizado no envio
  - SOURCE_SYSTEM – informação do sistema de origem da mensagem
  - MSG_TYPE – define o tipo de tecnologia utilizado (SMS/Email)
  - CONTROL_CARACTERES – define o controlo de caracteres
  - ORIGINATOR – define a origem da mensagem
  - DESTINATION – destinatário da mensagem
  - SUBJECT – assunto da mensagem (apenas Email)
  - REQUEST_DATA – data e hora do pedido
  - ORIGINAL_REQ_ID – ano de pedido da mensagem
  - PARAMETER_NAMES – nomes dos parâmetros a substituir no protótipo da mensagem.
  - PARAMETER_VALUE – valor de cada parâmetro a substituir no protótipo da mensagem.
  - PARAMETER_SEPARATOR – define o separador dos parâmetros.
  - RETRY_NUMBER – número de tentativas de envio da mensagem.
  - STATUS – define o estado atual de processamento da mensagem.
  - BUILT_MSG – define o corpo da mensagem a ser enviada.
  - ERROR_MSG – define a mensagem de erro originada no processamento
  - ERROR_CODE – define o código de erro de processamento
  - GATEWAY_REPLY_CODE – código retornado pelo *gateway* no envio
  - PROCESS_START_DATETIME – define início do processamento
  - PROCESS_END_DATETIME – data e hora do fim de processamento
  - API_CODE – código de identificação da mensagem pelo cliente.
  - SEGMENTS – número total de segmentos do corpo da mensagem SMS.

- **message_buffer_h** – armazena as informações das mensagens em estado final (enviados), contendo os mesmos atributos da tabela **message_buffer**.

- **configuração_smtp** – contem informação de configuração do servidor que será utilizado no envio de Email.
  - ID – identificação da configuração (Chave Primária).
  - SERVIDOR_IN – nome ou IP do servidor de recebimento.
  - SERVIDOR_OUT – nome ou IP do servidor de envio.
  - EMAIL – conta utilizado na autenticação e envio de email pelo servidor.
  - PASSWORD – senha de autenticação no servidor.
  - PORTA – porta do protocolo utilizado pelo servidor.
  - AUTH – define se é ou não utilizado autenticação.
  - SSL – define se existe ou não encriptação SSL na comunicação.
  - TLS – define se existe ou não encriptação TLS na comunicação.
  - FROM – define o nome do remetente da mensagem.
  - ACTIVO – define se a configuração está ativa no sistema.

- **gateway_sms** – contem informação de configuração do conta no fornecedor do serviço de SMS.
  - ID – chave primária da tabela.
  - NOME – nome do *gateway* (Twilio/ Clickatell).
  - CLIENTE_ID – código de identificação do utilizador no *gateway* Clickatell.
  - USER – nome de utilizador no *gateway*.
  - PASSWORD – senha de autenticação no *gateway*.
  - SMTP_API_ID – código de identificação do api do utilizador para uso do protocolo SMTP para envio de SMS pela Clickatell.
  - HTTP_SPI_ID – código de identificação da API HTTP do utilizador no *gateway* Clickatell.
  - FROM – número ou nome do remetente de SMS no *gateway*.
  - ACTIVO – determina se a configuração está ativado no sistema
  - URL_SEND – URL de envio das mensagens para o *gateway* por http.
  - MAIL_BOX_SEND – *email* de envio das mensagens por meio de SMTP.
  - ACCOUNT_SID – (user criptográfico) chave de autenticação no gateway Twilio.

- AUTHOTOKEN – (password criptográfico) chave de autenticação no *gateway* Twilio.
- CALLBACK – URL de retorno do relatório de envio para *gateway* Twilio, ou número 7 para gateway Clickatell.
- APPLICATION_SID – chave de identificação do CALLBACK no *gateway* Twilio.

- **server_control** – guarda informações do status atual do servido do sistema, e das configurações de notificação de alteração de status.
  - ID – chave primária da tabela.
  - STATUS – determina um estado do servidor Java do sistema.
  - NOTIFICAR_SMS – define se será enviado SMS a quando de alteração de *status* do servidor Java do serviço.
  - NOTIFICAR_EMAIL – define se será enviado *email* a quando da alteração de status do servidor.
  - TELEMOVEL – define o número de telemóvel que será notificado.
  - EMAIL – define o email que será notificado a quando da alteração.
  - LAST_UPD_USER – define o utilizador da última atualização da tabela.
  - LAST_UPD_DATE – define a data da última atualização da tabela.

- **user_limites** – contem informações de crédito do utilizador e de configuração dos limites de envio de alertas.
  - ID – chave primária da tabela.
  - USER_NAME – nome de identificação do utilizador.
  - USER_KEY – código do utilizador usado no envio de remoto de alertas.
  - MAX_SMS – número de SMS disponível na conta do utilizador.
  - MAX_EMAIL – numero de *Email* disponível na conta do utilizador.
  - MAX_SMS_DAY – máximo de envio de SMS permitido por dia.
  - MAX_EMAIL_DAY – máximo de envio de *Email* permitido por dia.
  - MAX_SMS_HOUR – máximo de SMS permitido por hora.
  - MAX_EMAIL_HOUR – máximo de *Email* permitido por hora.

3.2.3. Diagramas de Fluxo de Dados

O diagrama de fluxos de dados (DFD) é uma representação gráfica do "fluxo" de dados através de um sistema de informação, modelando seus aspetos de processo. Ele fornece apenas uma visão do sistema, a visão estruturada das funções, ou seja, o fluxo dos dados. Frequentemente os Diagramas de Fluxo de Dados são uma etapa preliminar usada para criar uma visão geral do sistema que pode posteriormente ser elaborado. Os DFDs também podem ser usados para a visualização do processamento de dados (design estruturado).

Um DFD mostra que tipo de informação entrará e sairá do sistema, para onde os dados virão e irão e onde os dados serão armazenados. Ele não mostra informação sobre a temporização dos processos ou informações se os processos irão operar em sequência ou em paralelo (o que é mostrado em um fluxograma).

A seguir é apresentado o DFD do 1º nível do sistema a desenvolver:

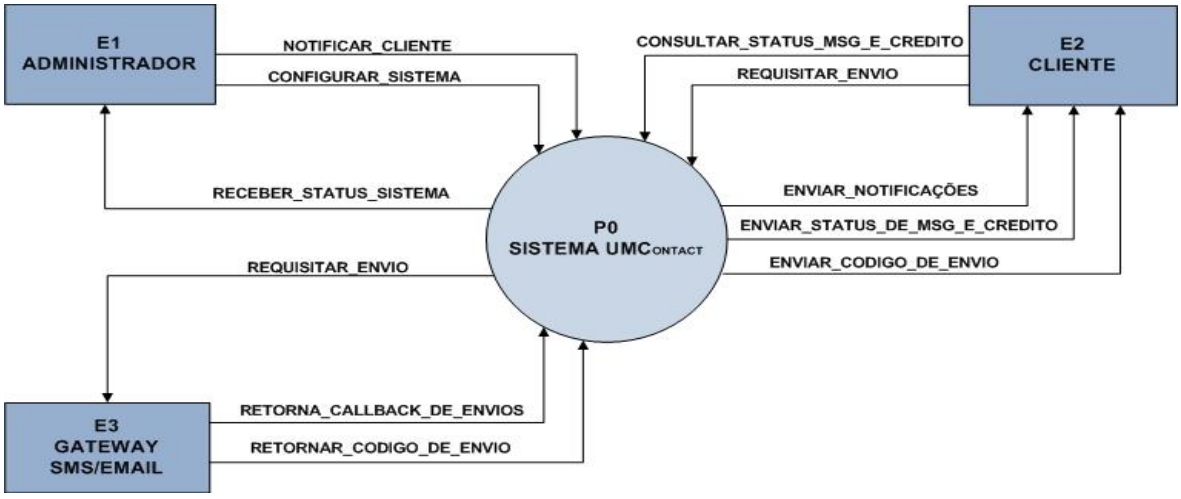

Figura 5 - DFD 1º nível

Para uma melhor compreensão do funcionamento do sistema, a seguir é apresentado o DFD do segundo nível:

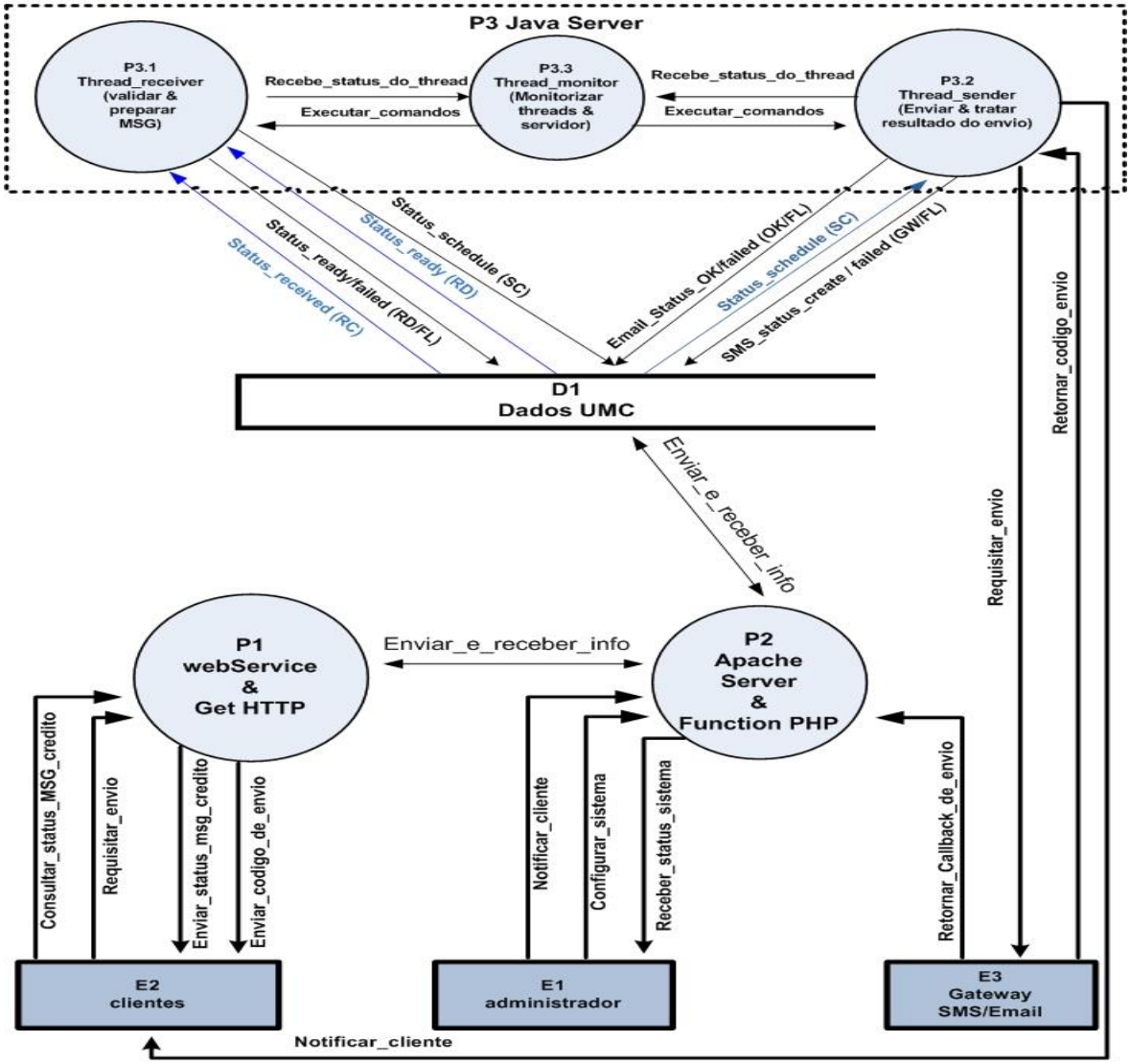

Figura 6 - DFD 2º nível

3.2.4. Diagrama de Caso de Uso do Sistema

Caso de uso é um documento narrativo que descreve a sequência de eventos de um ator que usa um sistema para completar um processo [Ivar Jacobson].

O sistema reconhecerá dois tipos de utilizadores:

- **O administrador -** terá acesso a todos os módulos do sistema, podendo assim administrar e configurar os diversos parâmetros do sistema, bem como administrar os utilizadores do sistema, e ainda monitorizar e efetuar comandos no servidor Java do sistema.

A seguir é apresentado o diagrama de Casos de uso do administrador do sistema:

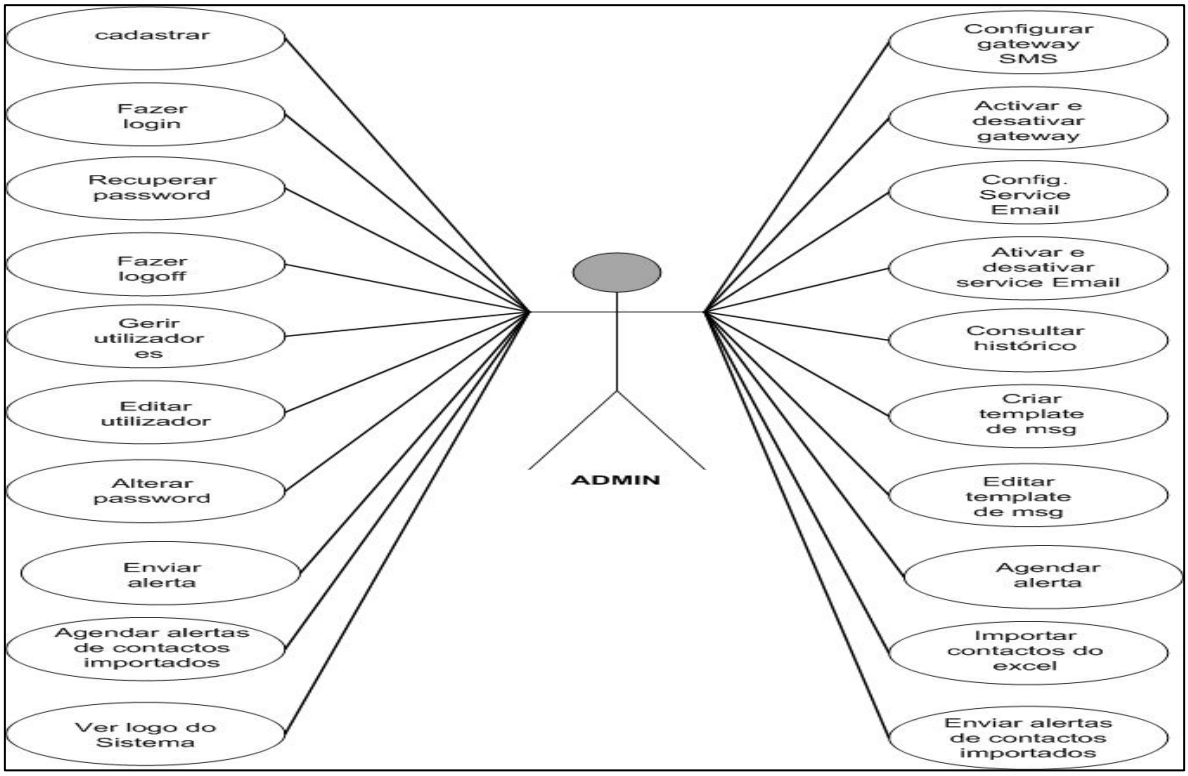

Figura 7 - Casos de uso do utilizador administrador

- **O cliente -** terá acesso limitado ao sistema, podendo apenas configurar a sua conta, controlar e configurar os seus serviços (limites de envios), criar e configurar os seus *templates*, importar contactos do Excel, enviar alertas e consultar o seu histórico de alertas.

De seguida é apresentado o diagrama de casos de uso dos clientes do sistema:

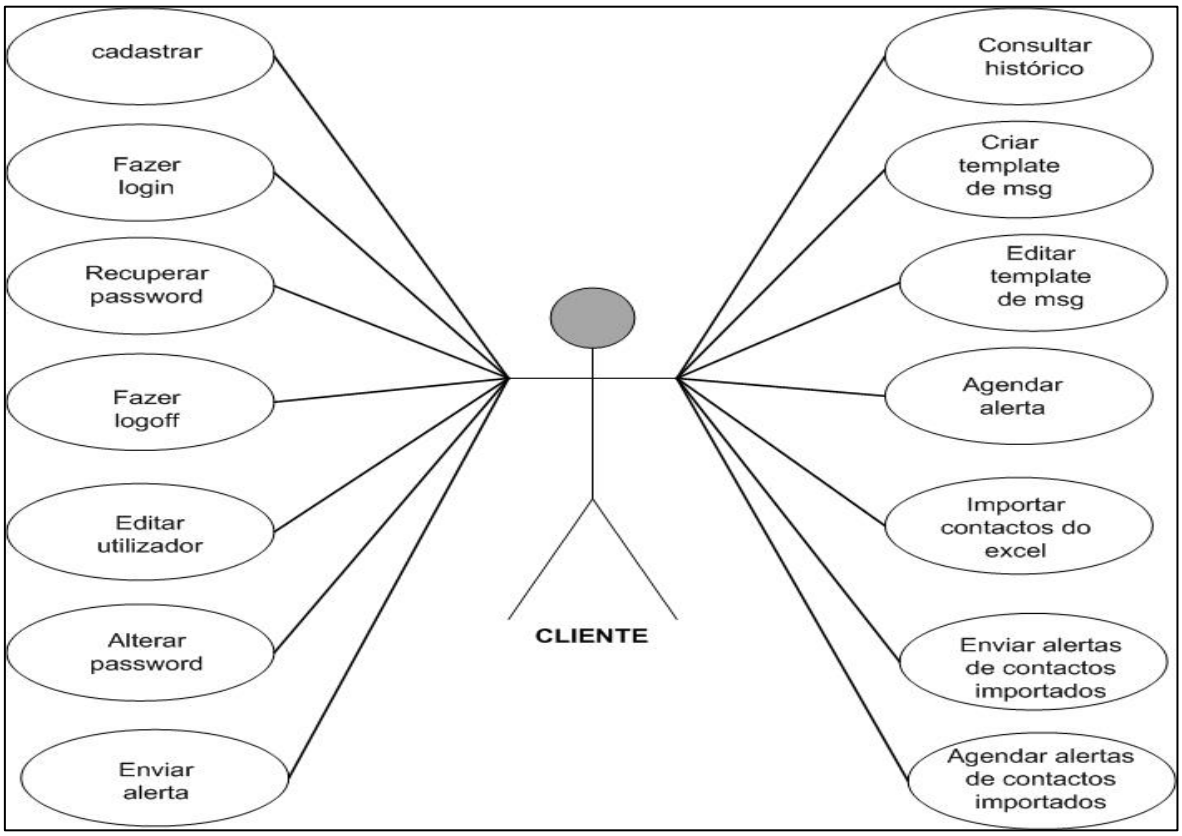

Figura 8 - Casos de uso do utilizador cliente

3.2.5. Diagrama de Sequência

Diagrama de sequência ou Diagrama de Sequência de Mensagens é um diagrama usado em UML (*Unified Modeling Language*), representando a sequência de processos

(mais especificamente, de mensagens passadas entre objetos) num programa de computador.

Um diagrama de sequência descreve a maneira como os grupos de objetos colaboram em algum comportamento ao longo do tempo.

3.2.5.1. Diagrama de Sequência – Registo de Utilizador

A gestão de utilizadores é de domínio do administrador do sistema, excetuando a ação de registo e atualização de utilizadores que poderá ser feito pelo próprio utilizador. Após o registo no sistema, o utilizador poderá a qualquer momento alterar o seu *password*, configurar os limites de envio, criar *templates* de mensagens e editar seus dados pessoais.

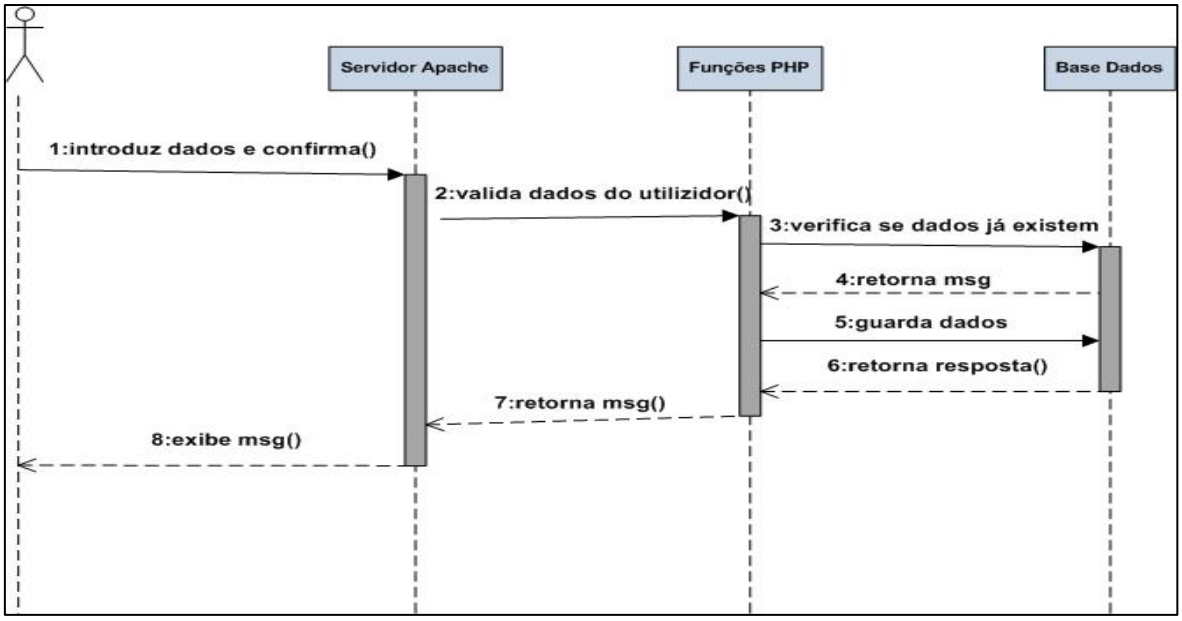

Figura 9 - Diagrama de sequência - registo de utilizador

3.2.5.2. Diagrama de Sequência – Login

O processo de login processa de forma idêntica a qualquer sistema. O utilizador informa as suas credenciais, estas passam pelo processo de validação na base de dados que em caso de sucesso redireciona o utilizador para a sua conta, ou informa a mensagem de erro adequado em caso de insucesso na validação.

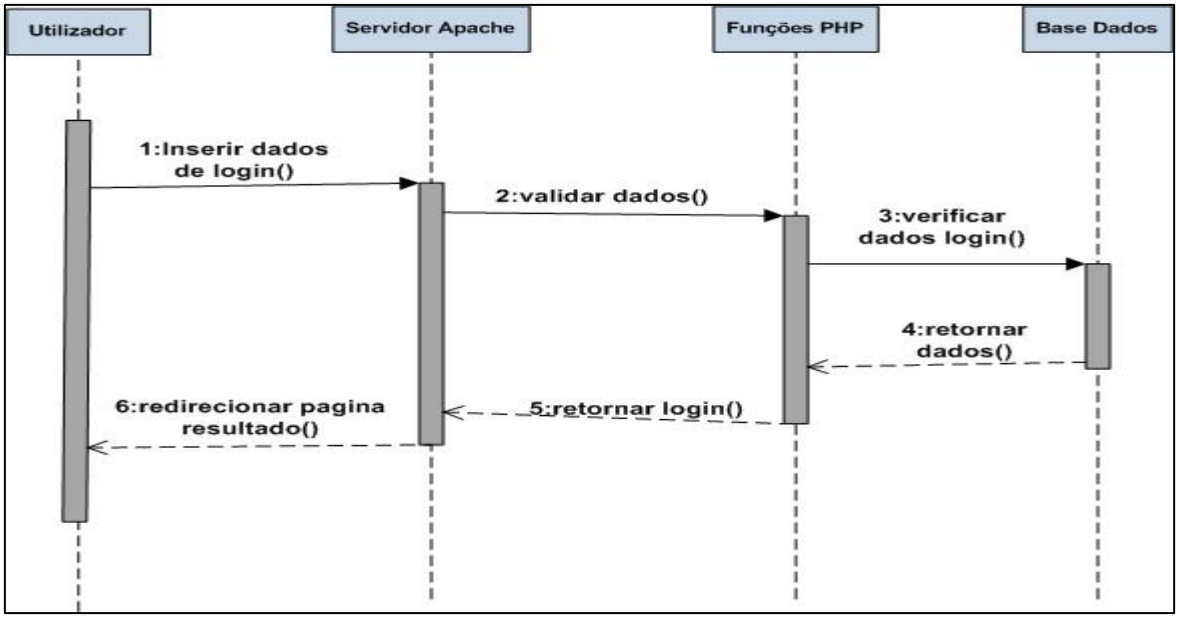

Figura 10 - Diagrama de sequência - login no sistema

3.2.5.3. Diagrama Sequência – Enviar Alerta

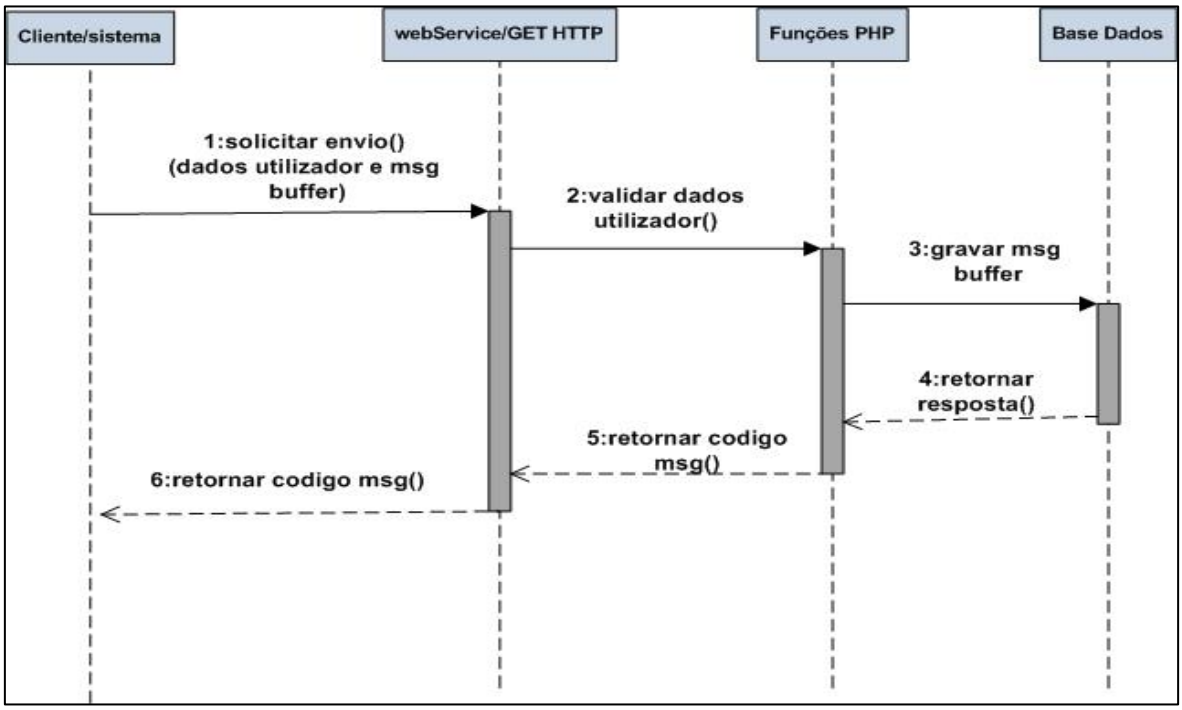

Figura 11 - Diagrama de sequência de envio de alertas

No envio de alertas a partir de sistema cliente tanto por *Web Service* como por HTTP POST e GET, o processo é similar. O sistema cliente faz a requisição que passa por um conjunto de validações através das funções PHP, guardando as informações em caso de sucesso. Para cada requisição bem-sucedida é retornado ao cliente um código que permite acompanhar a requisição junto ao sistema.

3.3. Problemas Encontrados e soluções

Ao longo do desenvolvimento do projeto, muitos foram os problemas encontrados, porém, com o aprofundar de pesquisas e estudo sobre e essas matérias, e com ajuda e colaboração do orientador e co-orientador, esses problemas foram gradualmente ultrapassados e tornando mesmo em importantes conhecimentos que doravante irão servir em futuros desenvolvimentos de projetos.

No desenvolvimento do servidor do sistema, que é responsável pelo envio das notificações, foi onde surgiu os maiores problemas, pelo facto de ser a parte mais importante de todo o sistema e por se tratar de um servidor. Qualquer problema surgido teria de ser bem analisado e encontrado a melhor solução.

Problema: **Entrada do servidor em *status* desconhecido** – problema em acompanhar o *status* do sistema após um longo período em execução. Sendo que muitas vez o sistema mesmo no *status Alive*, não desempenhava as suas funções de tratamento, validação e envio das alertas tanto SMS com correio eletrónico (*email*). Este problema surgia com muita frequência sempre após um longo período sem qualquer processamento de dados por parte do servidor.

Solução: **Introdução da API log4j** – com a introdução dessa API solucionou o problema acima descrito bem como outros problemas, uma vez que permitiu como o nome indica a criação de log do sistema. A API solucionou o problema através da criação de um log rotativo que produz informações tanto de depuração de erros do sistema bem como a saída de informações do *status* do servidor a cada instante (em milésimas de segundo), e por ser rotativo não é preciso preocupações sobre espaço de memória ocupado pelo log. Esse log permitiu criar tarefas *Runtime* para o servidor e assim impedir a sua entrada no *status* desconhecido.

Problema: **Monitorização dos Threads do servidor** – O servidor funciona a base de três *threads*, o ***receiver***, o ***sender*** e o ***monitoring***, sendo assim é importante o acompanhamento e monitorização de cada *thread* e tomar decisões em relação a diferentes situações ou erros produzidos pelas *threads*. O problema em si veio pelo pouco conhecimento sobre funcionamento de *threads*, o que a princípio dificultou e muito na criação da *thread monitoring* que como o nome já indica é a responsável pelo monitorização e tomada de decisões sobre as outas duas *threads* e do servidor.

Solução: **Criação da *class enumerable*** – na solução deste problema contou-se com a preciosa colaboração do co-orientador Flávio Fortes, tanto na indicação de excelentes tutoriais, na disponibilização de exemplos de código de monitorização de *threads* por ele desenvolvido, bem como na indicação de erros no código do servidor. Em paralelo também

foram realizados pesquisas em sites de fóruns de desenvolvedores Java como (GUJ). No final e como resposta a este problema foi desenvolvida um classe *enumerable* (classe num) em representação as diferentes estados possíveis de uma *threads* (*running*, *stoped*, *suspended*, *start*, *restart*), e assim melhorar a troca de informação entre as *threads receiver* e *sender* com a *thread monitoring*.

Na parte administrativa Web do sistema desenvolvido em PHP, o maior problema foi a falta de experiencia nessa linguagem de programação juntamente com o pouco conhecimento do *framework* utlizado no desenvolvimento (*Yii Framework*) que utiliza a nova conceção PHP Orientado a Objeto (PHP OO) disponibilizado e com um forte estrutura e suporte na nova versão PHP5.

Problema: Instalação de módulos – na instalação de diferentes módulos para dar ao sistema novas funcionalidades como gestão de menus, gestão de utilizadores e perfil, importação de conteúdos de ficheiros Excel, entre outros, sugiu diferentes erros ou situações que necessitaram de tratamentos especificos. O maior problema encontrado deveu-se ao elevado nível de separação de funcionalidades e regras do *framework* em si, derivado pela excelente arquitetura do Yii (arquitetura MVC).

Solução: Para a resolução do problema de intalação de modulos foi necessario enrequecer conhecimentos no funcionamento do *framework* com a ajuda do co-orientador Flávio Fortes e do colega estagiario Emanuel Vieira que sempre se mostraram disponíveis. Por outro lado pesquisas no projeto SIGA (Sistema Integrado de Gestão Académico) que está sendo desenvolvido na Universidade do Mindelo, permitiu-me também enriquecer conhecimentos na instalação de módulos no *framework*, bem como no conhecimento geral do *framework* .

3.4. Testes Realizados

O objetivo deste subcapítulo é a descrição do plano de testes realizado no sistema, de forma a percorrer todos os pontos do sistema e assim garantir o seu normal funcionamento. Os módulos devem ser testados para garantir que as informações fluem para dentro e fora como esperado.

De referir que foram realizados avultados teste de *stress* perante o modulo de SMS, a quando do envio de notificações aos novos estudantes do 1ºano da Universidade do Mindelo sobre diferentes assuntos. Estes testes estiveram na origem da utilização de um novo *gateway* de serviço SMS para o sistema, uma vez que os preços anunciados pelo *gateway* até ali utilizado não correspondiam a realidade praticada, ou seja, anunciava um preço e no momento de envio o valor descontado era bem superior ao anunciado.

3.4.1. Testes de Caixa Branca

Esta técnica avalia o comportamento interno de cada módulo, operando diretamente sobre o código fonte do *software* para avaliar aspetos como condições lógicas, teste de ciclos, teste de fluxos de dados e testes de caminhos lógicos.

Para esta técnica deve-se ter acesso ao código fonte do *software*, permitindo a separação de módulos e até mesmo métodos ou funções do sistema, para assim poder verificar o comportamento individual de cada um.

Esta técnica foi bastante utilizada e os testes realizados basearam-se sobretudo sobre fluxo de informações, ou seja verificar o funcionamento a quando da entrada e saída de informações em cada módulo. Em que o sistema foi submetido a informações corretas e erradas, verificando a resposta a cada situação. Bastante utilizada no momento de codificação de cada módulo do servidor, esta técnica permitiu a correção e melhorias em diferentes métodos utilizados pelas *threads* de envio e recebimento de alertas.

3.4.2. Teste de Caixa Preta

Esta técnica é utilizada quando o teste de um *software* assenta-se apenas e exclusivamente sobre o input e output do mesmo, ignorando o seu comportamento interno. O objetivo aqui é a execução do *software* com determinado input, e o output (saída) é comparado com o valor esperado.

Esta técnica foi a mais utilizada na fase de teste do servidor sendo muito útil na detenção de alguns bugs relacionados com a fase de validação dos *buffers* de mensagens e fase de tratamento do resultado de envio. Está técnica também está por trás do resultado final obtido sob o *thread* de monitoramento do servidor, uma vez que permitiu a captura e correção de bugs relacionados com uma das funcionalidades do *thread* de monitoramento do servidor, a funcionalidade de restar das outras duas *threads* em caso de falha pré-determinadas.

3.4.3. Teste de Segurança

A segurança é uma característica fundamental em qualquer sistema principalmente quando se trata de um sistema de interface Web, tanto para garantir a integridade do próprio sistema como também de informações muitas vezes sigilosas.

Neste contexto que a parte Web do sistema foi fortemente testado com diferentes privilégios (visitante, cliente e administrador) nas diferentes ações, e comparando-os com respostas ou situações previamente determinados.

Uma vez que o sistema opera sob URL amigáveis, os testes de acesso foram fortemente elaborados para garantir o máximo de segurança do sistema. Em ambas situações, os testes responderam de forma esperada, mostrando a mensagem correta para cada caso.

3.5. Resultados dos Diferentes Teste

Durante toda a fase de desenvolvimento e teste, todo o sistema (servidor apache, servidor UMC, base dados MySQL) esteve hospedado numa mesma máquina, rodado sobre o sistema operativo *Ubunto Server* v14.04 com CPU *Intel Pentium Dual* de 2403MHz e 2GB de RAM. Sobre ele roda um *XampServer* versão 1.8.3 que suporta o servidor apache e a base de dados MySQL usados no projeto.

Para os testes da parte administrativa do projeto foi criado dois utilizadores com os dois *perfis* suportados pelo sistema (administrador e cliente). Foram realizados teste de ação sem efetuar login no sistema, e está respondeu da forma desejada. Fez-se também testes com utilizador inválido e senha inválida e em ambas situações o sistema respondeu da forma esperada e com a mensagem apropriada.

O acesso a URL de acordo com o perfil de utilizador ocorreu de forma esperada não demostrando nenhuma falha.

O Web Service e GET PHP de acesso pública, utilizados na conexão dos clientes com o sistema também foram testados, em diferentes situações e, em ambas, a resposta foi a esperada.

A amigabilidade da interface Web do sistema foi outro teste realizado, e sendo os potenciais utilizadores da interface, individuos com perfis técnicos, os utilizadores responderam positivamente sobre a facilidade de uso da interface.

O servidor também foi fortemente testado, e por tratar os *inputs* e outputs através de diferentes STATUS (RC - recebido; RD - validado; SC - agendado; FL - falha; GW - entregue ao *gateway*), os testes aqui centraram-se na tentativa de burlar os Status através de informações inválidas. E em ambas as situações o servidor respondeu de forma esperada, com o código e mensagem de erro esperado.

A *Thread receiver* que trata do processo de validação e preparação das alertas aquando da sua chegada no sistema, foi testada através da técnica de caixa branca, em que se tentou burlar de toda a forma o seu processo. A resposta foi sempre a esperada, colocando a alerta no *status* de FL com o respetivo código de erro e mensagem de erro apropriado.

Da mesma forma a *thread sender* responsável pelo processo de envio e tratamento do resultado de envio, foi testado com a mesma técnica, o que permitiu corrigir algumas exceções (*exception*) produzidas em algumas situações.

O processo de agendamento de alerta mostrou algumas falhas e prontamente foram ajustados de acordo com as ideias da sua concepção.

Após a correção do problema de operabilidade inicialmente detetada no servidor, corrigida com a implementação do processo de log do sistema, o servidor tem demostrado excelentes níveis de produção, e já com mais de meses em teste não demostrou qualquer problema que pudesse ser aqui descrito.

4. PROTÓTIPO DESENVOLVIDO

Neste capítulo será abordado o protótipo proposto para este projeto, bem como a descrição das principais funcionalidades desenvolvidos nele.

4.1. Descrição Geral do Sistema

A seguir é demostrada as principais funcionalidades do sistema, em que se destacam a funcionalidade de autenticação, recuperação de *password*, disponibilização de Menus, importação de contactos móveis do Excel, de *templates* de mensagens, POST HTTP, entre outros.

4.1.1. Autenticação

Os utilizadores a considerar no sistema são os clientes e os administradores. A autenticação é feita através do *email* ou nome de utilizador e a *password*, sendo estas pessoais e intransmissíveis a outrem por questões de segurança e integridade dos dados da conta do utilizador e do próprio sistema.

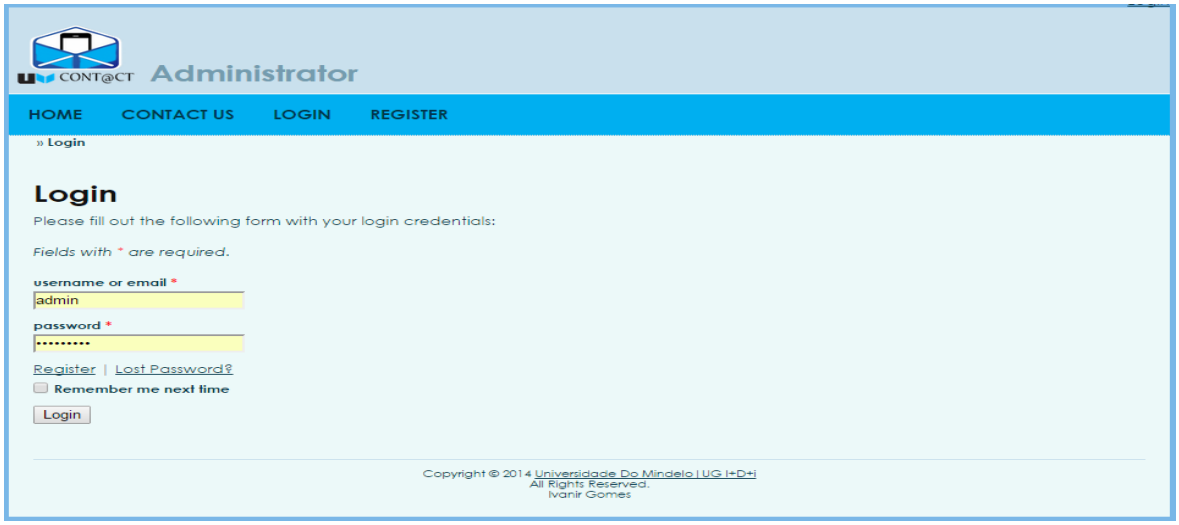

Figura 12 - Autenticação no sistema

4.1.2. Recuperação de Password

A recuperação de acesso no sistema é feita através do *username* ou *email* do utilizador em que é enviado um email ao utilizador com instruções de recuperação.

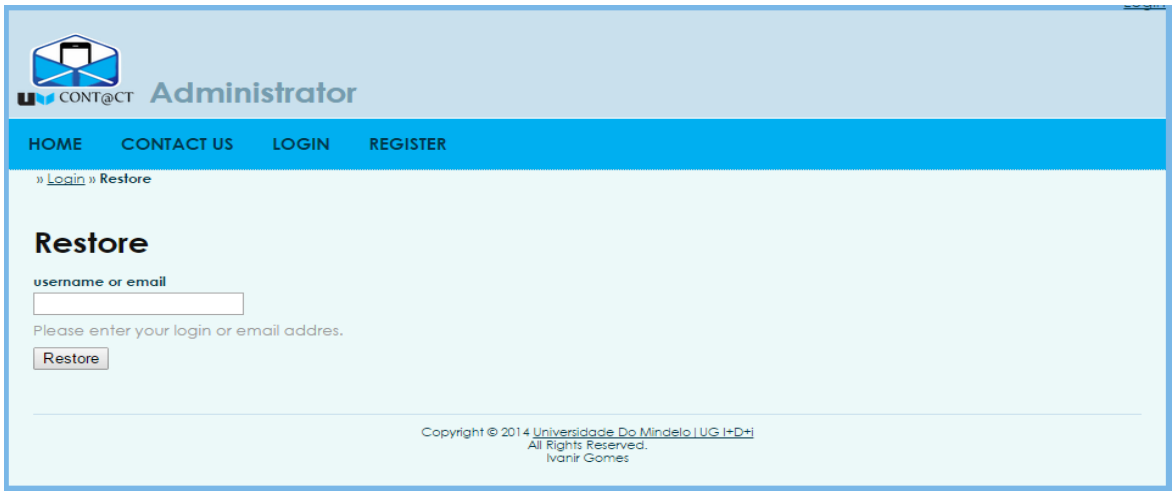

Figura 13 - Recuperação de *password* no sistema

O *link* enviado ao utilizador permite que este define um novo password para a sua conta.

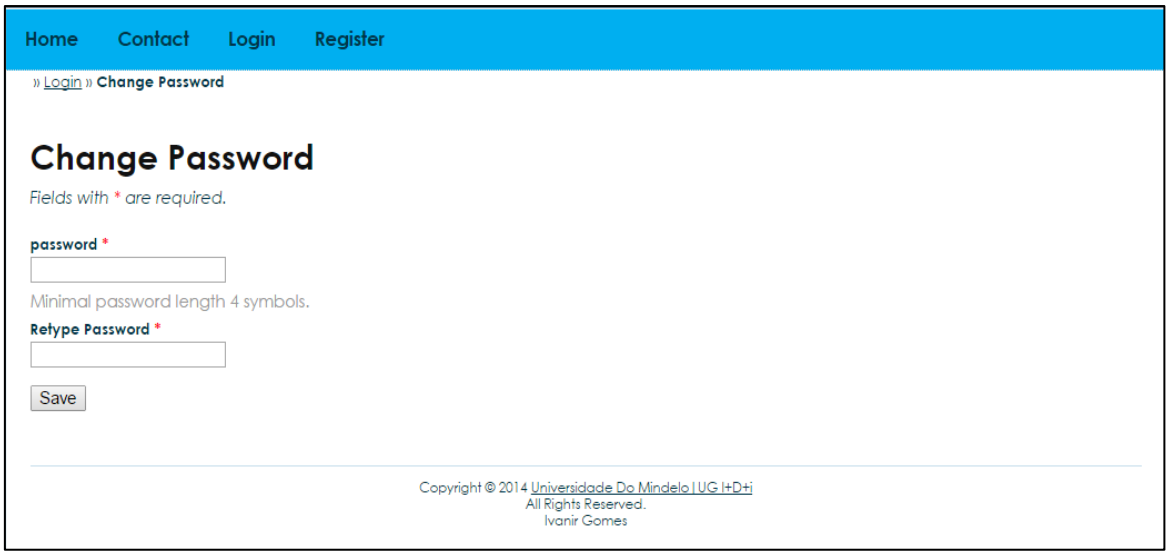

Figura 14 - Redefinição de *pasword*

4.1.3. Painel de Menu

A geração de menus é importante em qualquer sistema informático, visto que facilita a navegação do utilizador, permitindo uma maior interatividade entre o sistema e o utilizador.

O sistema disponibiliza um módulo que permite aos utilizadores administradores gerir os menus do sistema.

O sistema disponibiliza os seguintes menus para navegadores visitantes:

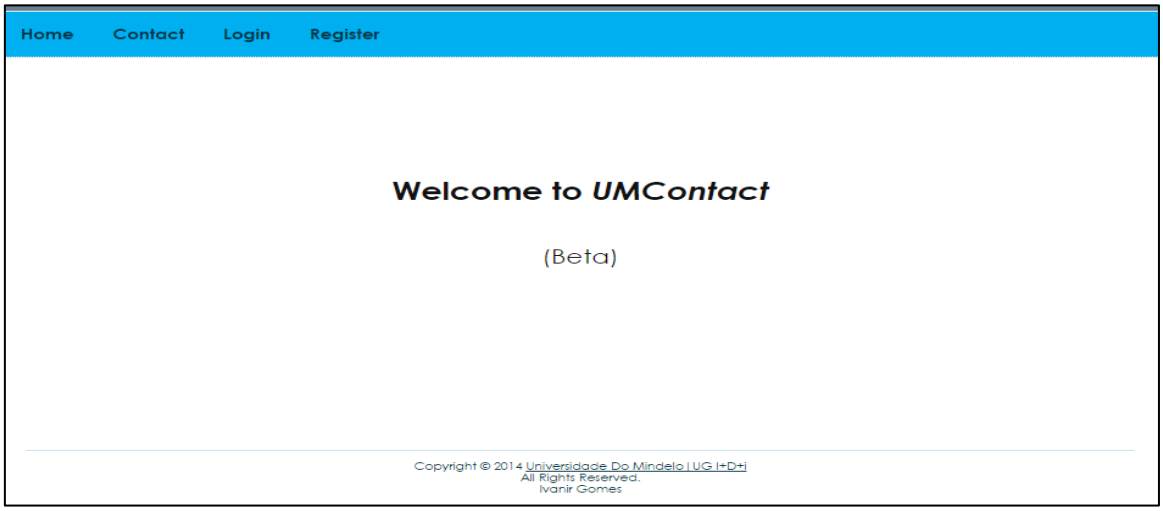

Figura 15 - Menus do utilizador visitante

Após efetuar o login, são disponibilizados os seguintes menus:

- HOME
- *MESSAGING*
  - MSG *Buffer*
  - MSG *History*
  - MSG *Template*
- EXCEL CONTACT

- SETTING
  - SMS
  - *Email*
  - *Server Control*
  - Gerir Menus
- LIMITS
- CONTACT US
- ACCOUNT
  - View Perfil
  - Edit Perfil
  - Change Password
  - Logout

A seguir é apresentado a figura que demostra a disponibilização de menus.

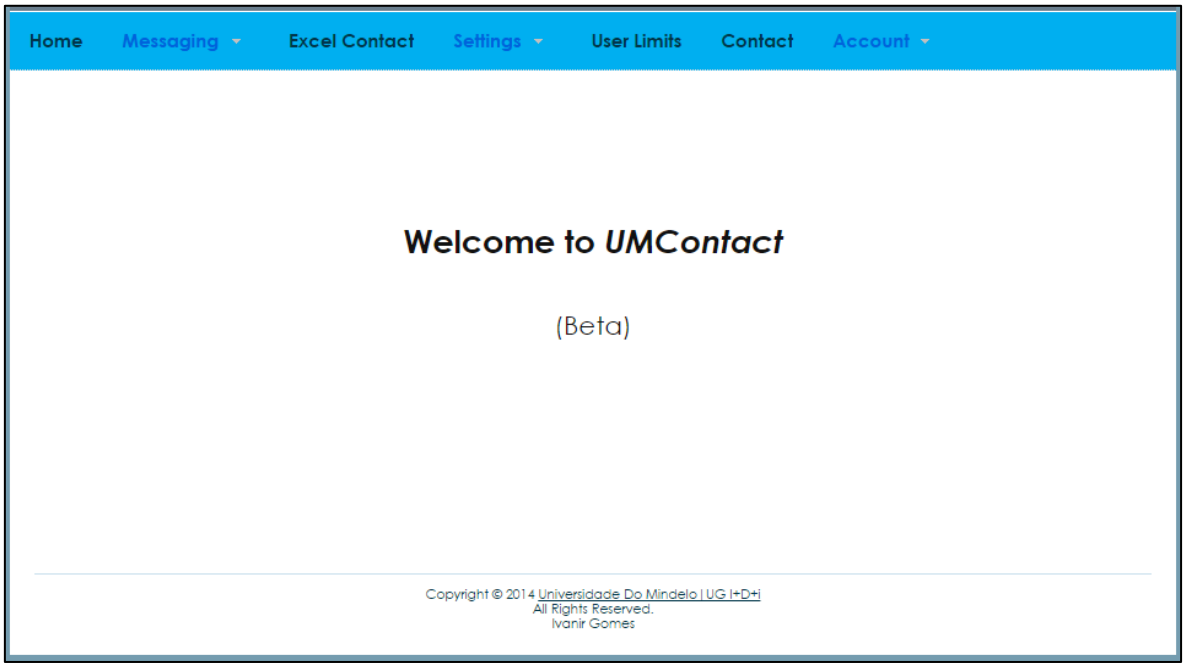

Figura 16 - Menus para utilizadores após login

4.1.4. Criação de templates

Todo o processo de envio de notificações baseia-se em *templates* de mensagens. Os *templates* contém diversas informações que são passadas para os *buffers* durante o processo de validação de buffer. O *template* decide a tecnologia a utilizar no envio, o modo de envio (agendado ou instantâneo), a hora e dia de envio, o utilizador da mensagem, a utilização ou não do controlo de caracteres nas mensagens SMS, entre outros.

A seguir é apresentado um exemplo de criação de um *template* SMS instantâneo.

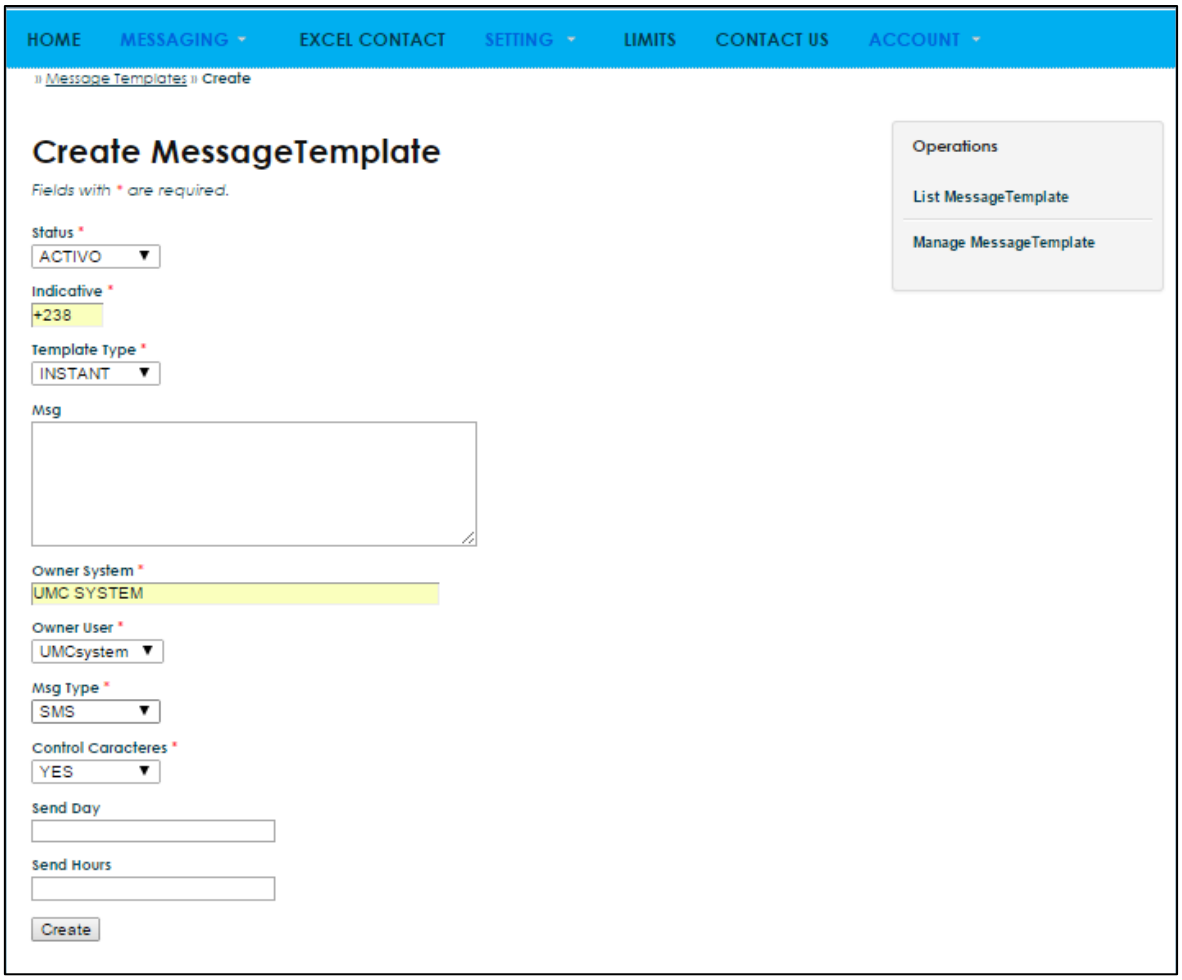

Figura 17 - Criação de *template* tipo instantâneo

Para uma melhor gestão de crédito do utilizador, este pode optar por escolher o **controlo de caracteres** para os seus *templates*. Este mecanismo permite que nas

mensagens SMS enviadas utilizando o *template*, sejam feitas as substituições dos caracteres acentuados (ç, á, ã, â, etc.) por caracteres normais (c, a), diminuindo o número de segmentos para uma mensagem SMS.

4.1.5. Agendamento de Alertas

Uma outra funcionalidade do sistema é o agendamento de alerta. Este permite que os clientes criam alertas que podem ser armazenadas no sistema, com desigançao da da hora certa para ser enviadas. O processo de agendamento depende do *template* utilizado na criação das alertas, pois o agendamento dá-se pela simples indicação no *template*, o dia e a hora que as mensagens criadas pelo *template* serão enviadas. O dia e a hora poderá ser tanto singular como um intervalo de dia e hora.

A seguir é apresentado um exemplo de um *template* que faz agendamento das mensagens:

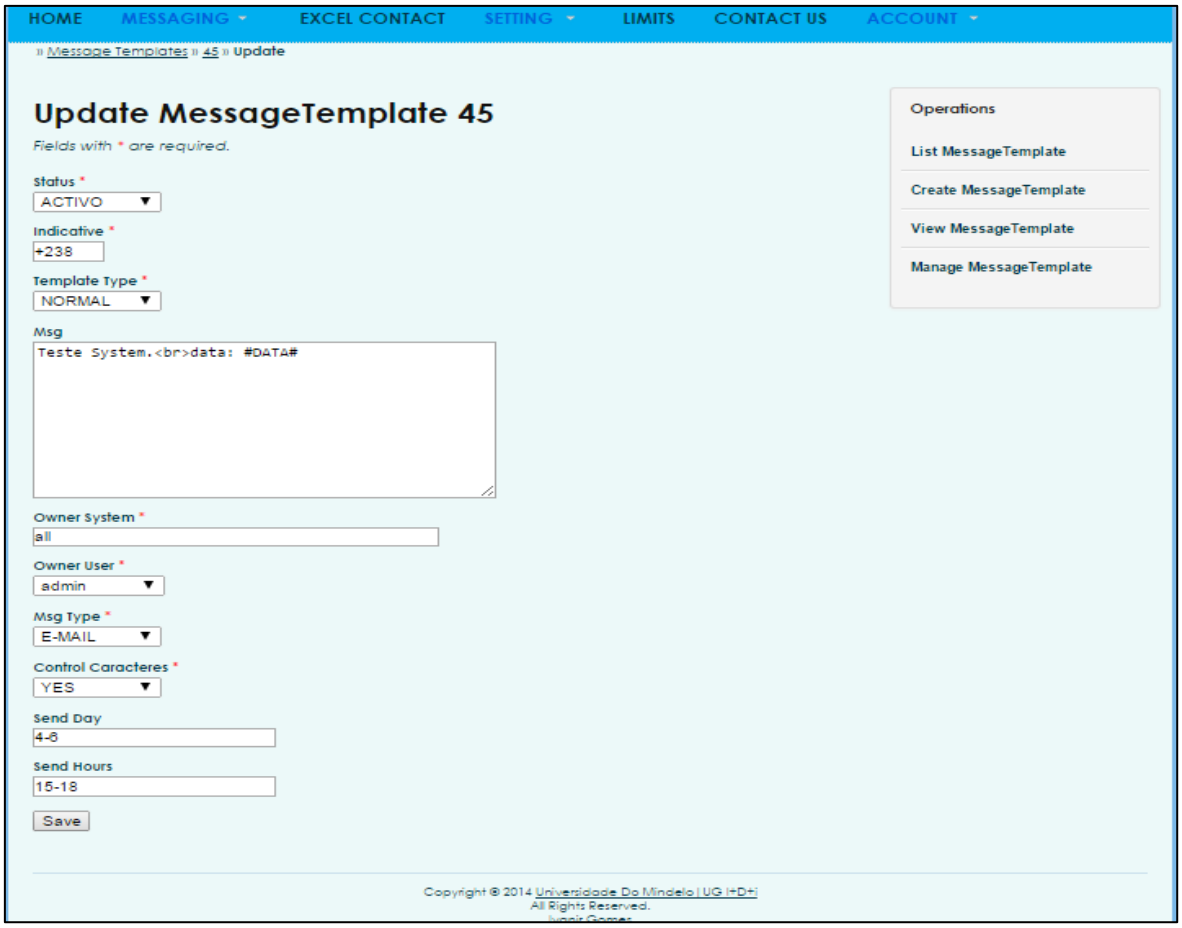

Figura 18 - Criação de template agendado

No *template* exemplificado acima, todas as mensagens originadas por ele serão agendadas para o intervalo de quarta-feira (4ª) à sexta-feira (6ª) das 15 horas às 18 horas. Sendo que qualquer pedido de mensagem deste *template* que chegue fora deste intervalo, permanece no sistema até que se chegue a um tempo que respeite esse intervalo.

4.1.6. Importação de Contactos

O sistema fornece um módulo que permite aos clientes do sistema importar contactos telemóveis de um documento Excel, para futuros envios de mensagens. Os grupos de contactos são identificados por uma chave, a ser utilizada no momento de envio das mensagens. Como requisito da importação é exigida a presença de uma coluna com

cabeçalho "telemovel" ou "telemóvel" sem sensibilidade de caracteres. Na presença de uma coluna "Nome" esta será também importada, sendo o seu valor *null* caso contrário. Também o sistema permite ao utilizador a possibilidade de visualizar o conteúdo antes de uma importação.

A seguir é apresentado uma visualização do conteúdo de um documento Excel.

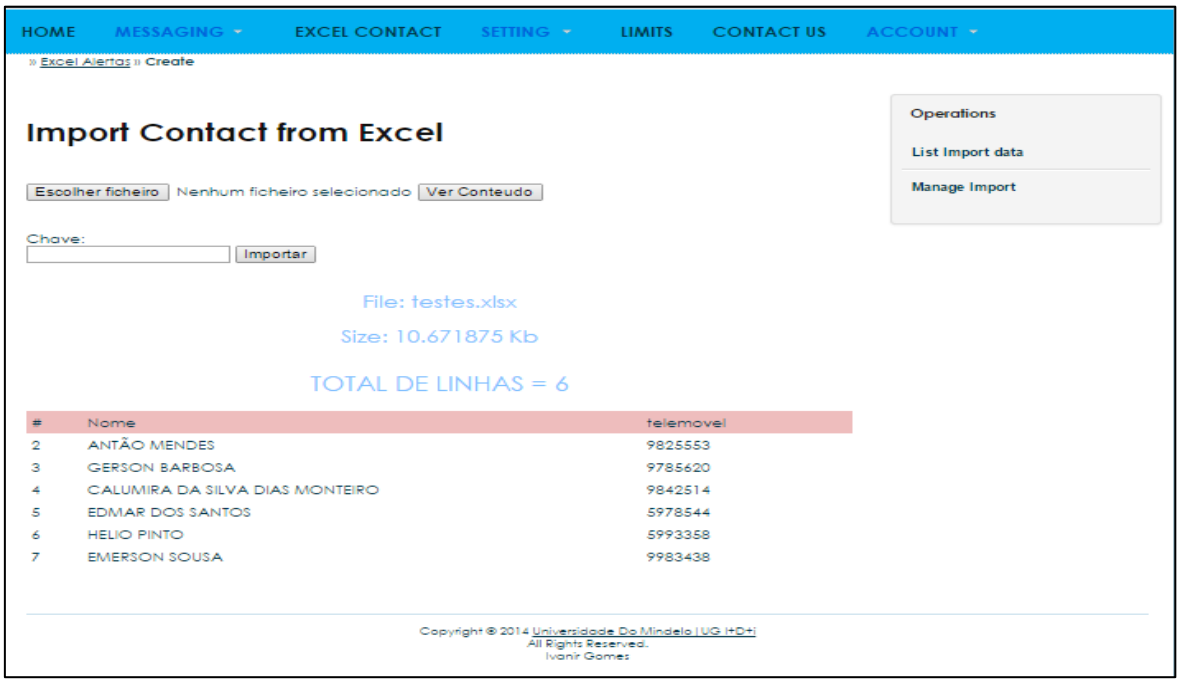

Figura 19 – Visualização de conteúdos de ficheiros do Excel

Após a visualização, o utilizador pode importar os contactos existentes no ficheiro. A importação é afeta ao conteúdo do ficheiro e não do ficheiro em si.

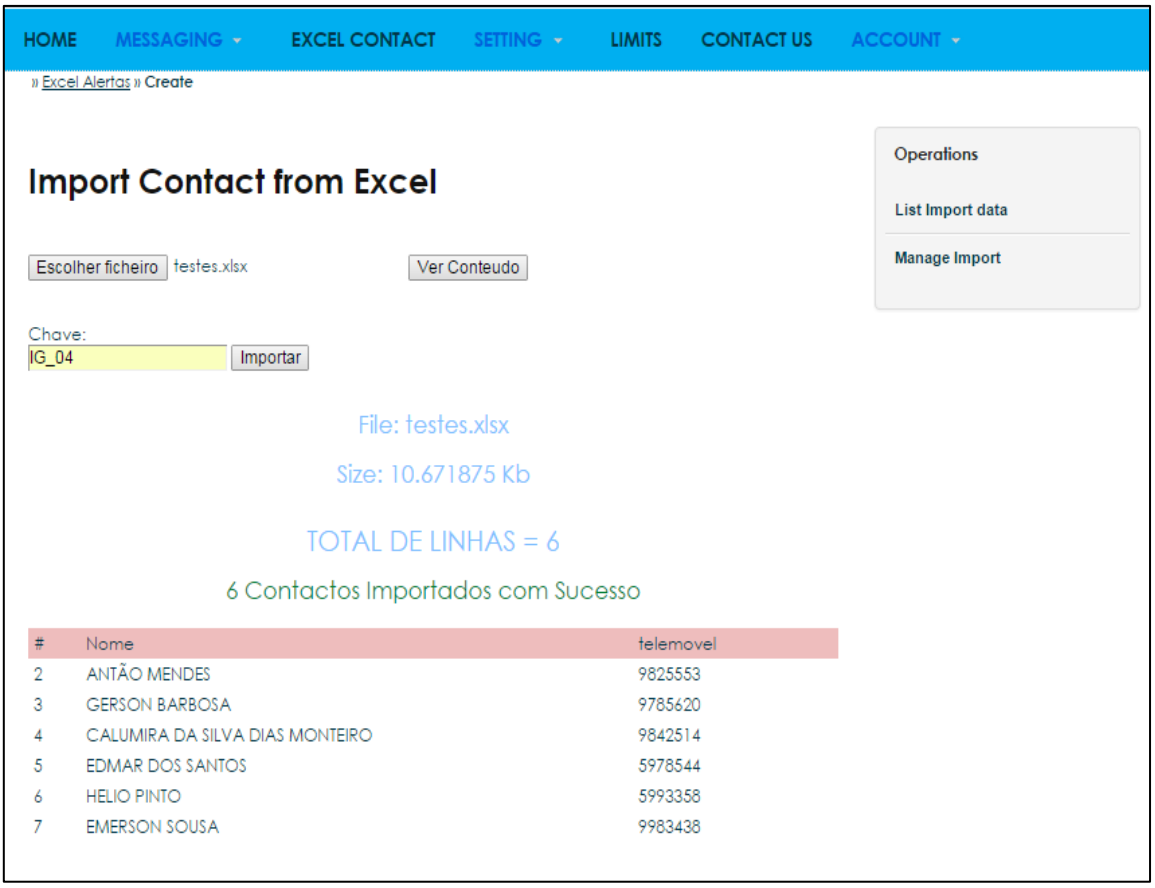

Figura 20 - Importação de contactos do Excel

Ao utilizar uma chave anteriormente utilizada numa importação, os contactos dessa chave são atualizados, ou seja, são acrescentados apenas os contactos que ainda não existiam nessa chave, sendo os que já existiam nessa chave ignorados na importação.

A seguir é apresentado uma importação para uma chave utilizada anteriormente e com contactos já existentes:

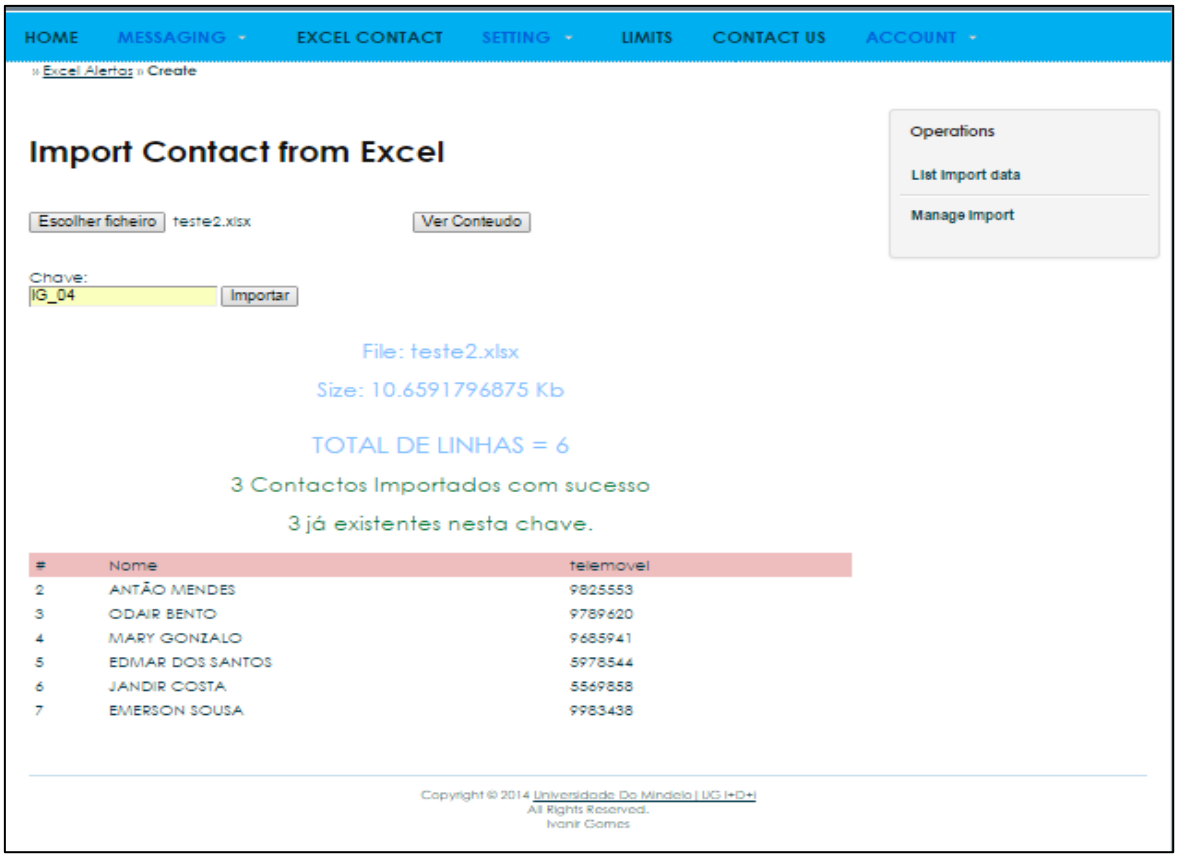

Figura 21 - Atualização de uma chave de importação

4.1.7. Configuração de gateway SMS

Pela parte administrativa, os utilizadores administradores podem a qualquer momento configurar ou alterar as configurações de comunicação com os *gateways* que fornecem o serviço de SMS. Para isso esses utilizadores devem aceder ao menu SETTING onde para além das configurações de SMS eles podem configurar diversos parâmetros do sistema.

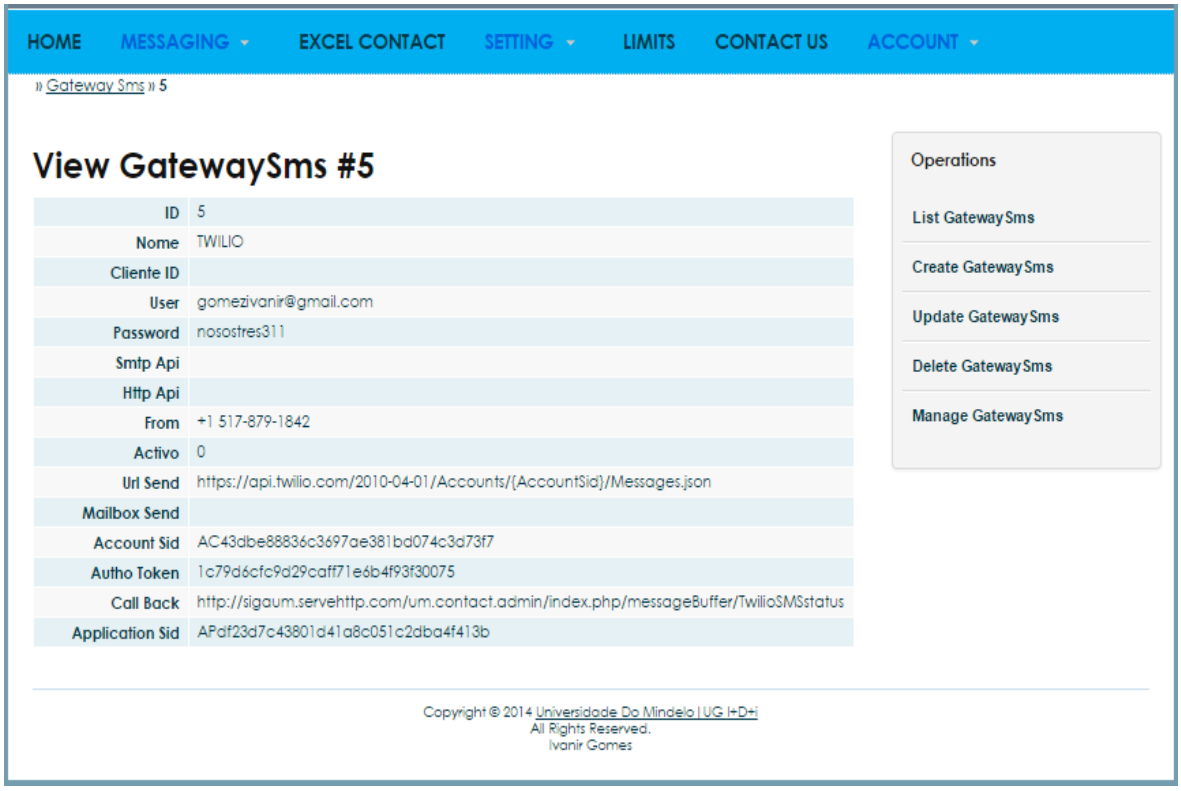

Figura 22 - Configuração de gateway SMS

4.1.8. Configuração do Servidor SMTP

A comunicação UMC com o servidor de *email* é outro ponto configurável pela parte administrativa do sistema. Projetado para utilizar qualquer servidor publico dos mais conhecidos como *Gmail* e *Outlook* (Hotmail), o sistema também poderá ser configurado para servidores privados como é o caso do servidor da Universidade de Mindelo.

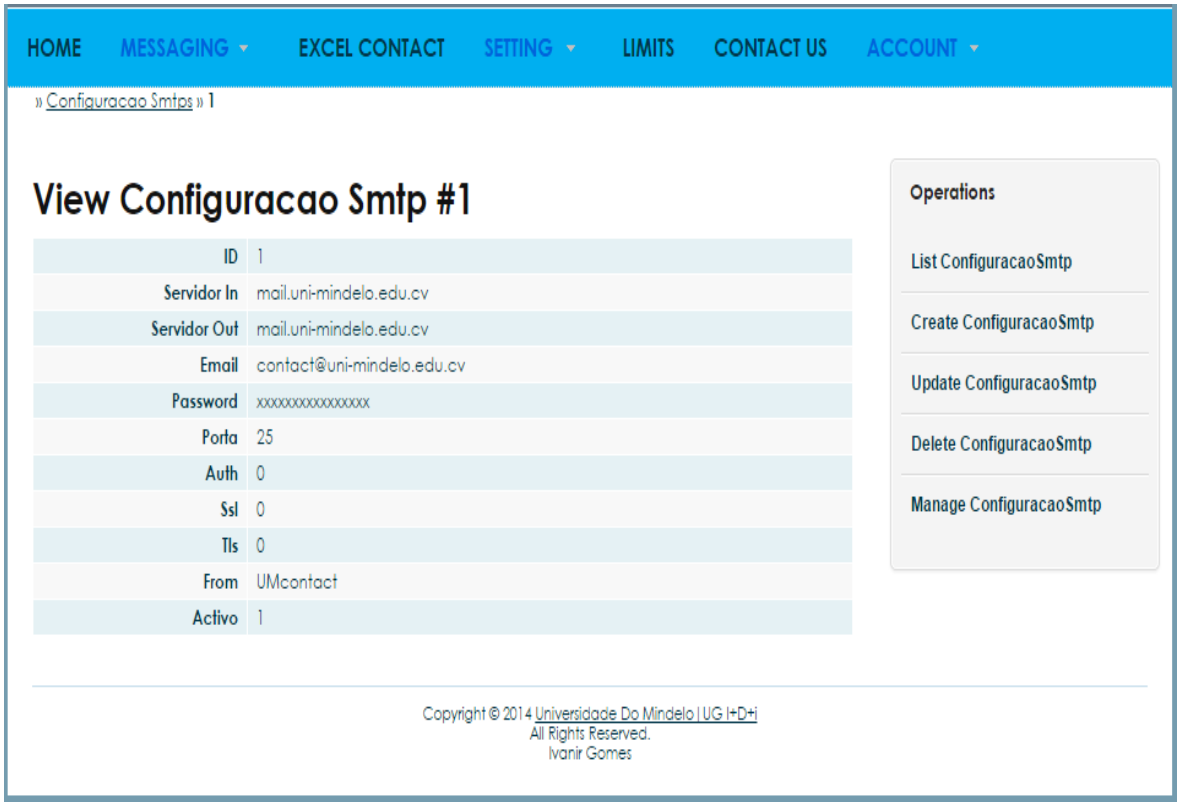

Figura 23 - Configuração da comunicação SMTP

4.1.9. Controlo do Servidor UMC

Para uma rápida intervenção no servidor UMC (servidor Java), é disponibilizado ao administrador do sistema uma forma de executar diferentes comandos pela parte Web do sistema. Os comandos START, RESTART, SUSPEND, KILL, RUNNING são programados para executar as diferentes ações sugeridos pelos nomes.

Em caso de alteração de estado as opções de alerta por SMS e *Email* para o administrador do sistema, é disponibilizado.

O comando START é o único executado via script, para isso foi criado um script que faz o lançamento do sistema. O script é executado via Web através da parte administrativa.

Ao executar o script, este faz a verificação da existência ou não do processo em execução para impedir duplicação de processo. Este mesmo script é utilizado para lançar o sistema no momento que o servidor de hospedagem for iniciado.

Os restantes comandos são executados via código fonte do servidor Java, sendo as configurações armazenadas na base de dados e posteriormente lidas e executadas pelo servidor Java.

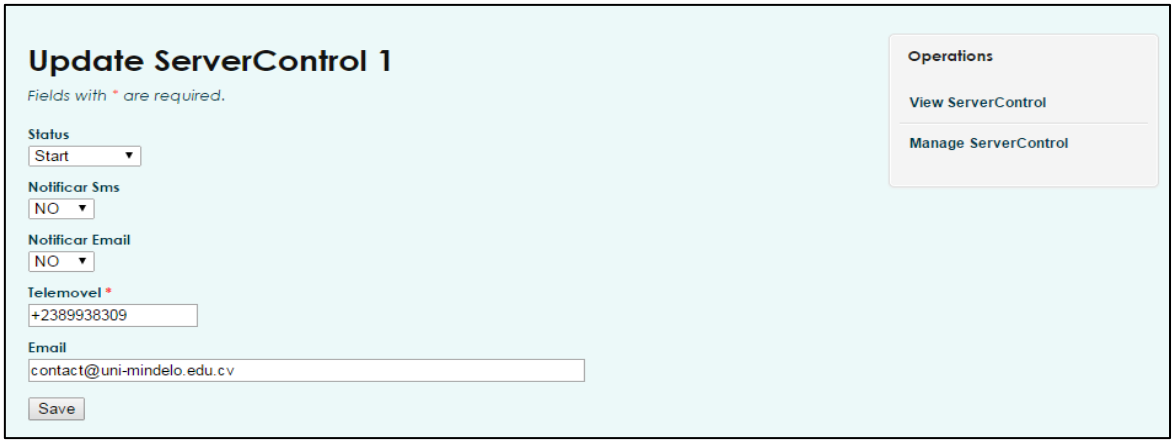

Figura 24 - Formulário de Controlo do servidor Java

O script utilizado para o comando START é demostrado a seguir:

```
#!/bin/sh

################################################################################
# Arranque do servidor
################################################################################

PID=`ps -ef | egrep -v 'grep|color' | grep um_contact_server | awk '{print $2}'`


if [ -z "$PID" ]; then

     cd /home/siga/UMcontactServer/bin/
     nohup java -jar /home/siga/UMcontactServer/bin/um_contact_server.jar $* >/dev/null 2>&1 &
     echo "um_contact server started!"
else
    echo "um_contact server is already running with pid: "$PID
fi
```

Figura 25 - Script de execução do servidor Java

4.1.10. Configuração de Menus

Para uma rápida configuração de menus, o sistema conta com um módulo que permite ao administrador, a qualquer momento criar, editar, remover qualquer item de menu do sistema, sem necessidade de entrar pelo código do projeto.

Este módulo permite tomar decisões como ativar ou desativar, visibilidade do menu, parentesco entre menus, opção de abrir ou não noutra janela, etc. Para além disso permite localizar a barra de menu num local específico da página.

A figura a seguir mostra a imagem desse módulo do sistema:

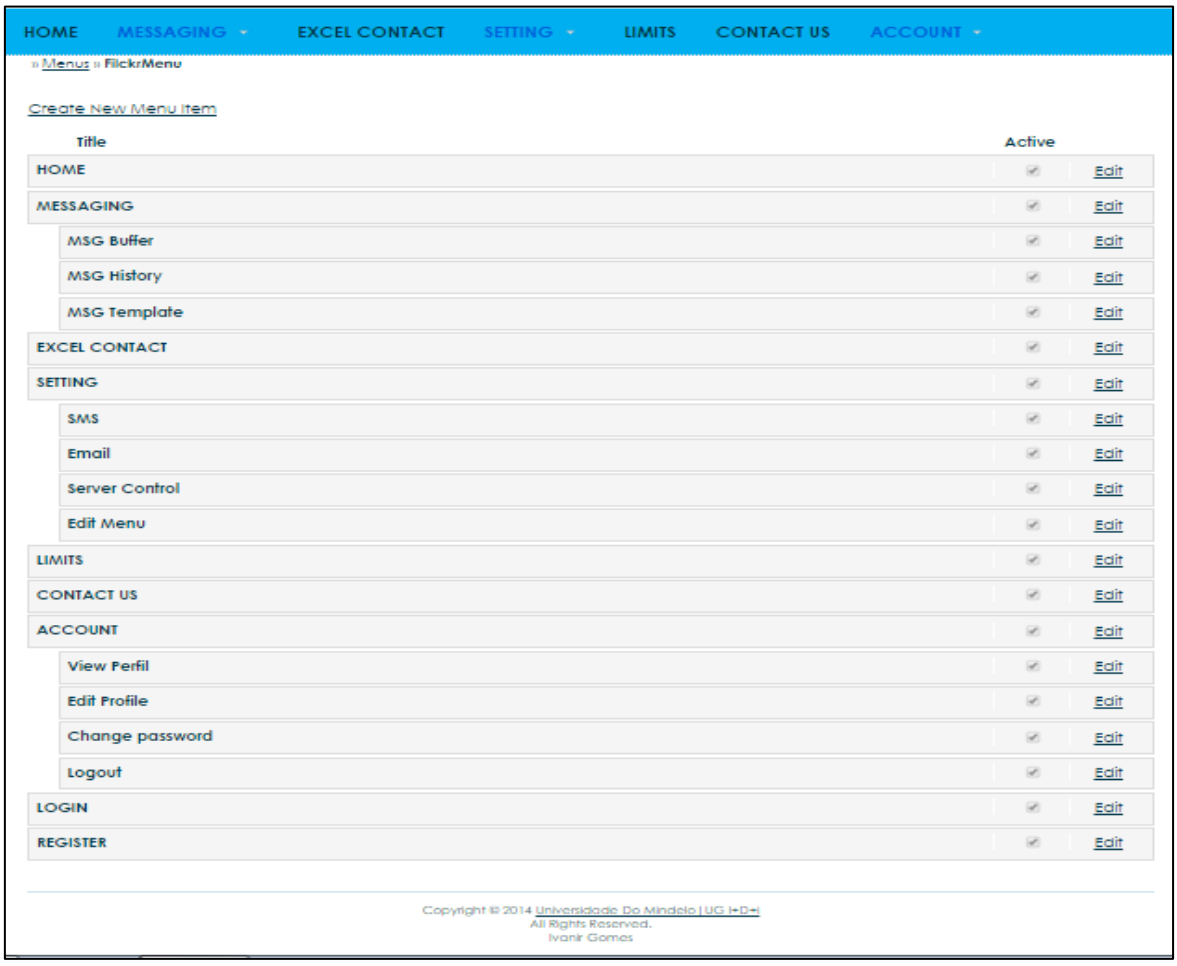

Figura 26 - Edição de Menus do Sistema

4.2. Envio de Mensagens

O envio de mensagens SMS e *Email* podem ser feitas de várias formas. O utilizador pode optar por criar uma conexão entre a sua aplicação e a UMCont@ct utilizando para tal os POST e GET HTTP ou o *Web Service* disponibilizado, ou mesmo utilizar os métodos de envio disponibilizados no formulário de envio na sua página UMC.

4.2.1. Envio de SMS para contactos importados

Para o envio de mensagens aos contactos importados do Excel, o sistema disponibiliza um formulário que permite ao utilizador processar o envio aos contactos importados em cada chave. No processo é necessário a chave de importação que identifica os contactos, o *template* de SMS a utilizar e a mensagens que se pretende enviar.

A seguir é apresentado o formulário disponibilizado para o efeito:

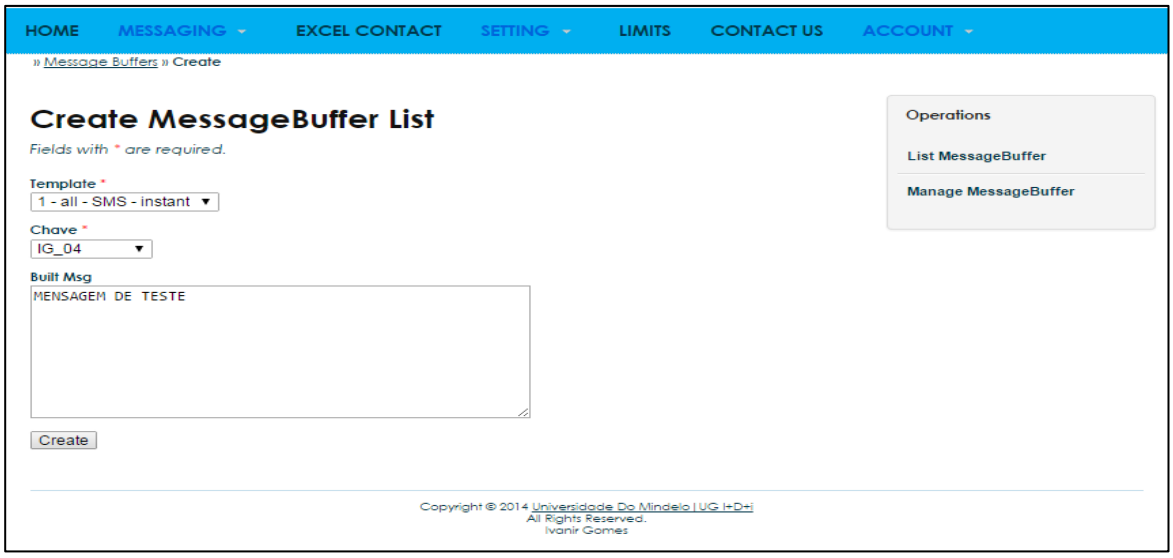

Figura 27 - Formulário de envio de SMS para contactos importados

4.2.2. Envio de mensagens por Request http

Para envio de mensagens pelo protocolo HTTP, é disponibilizado aos clientes dois métodos de acesso por GET e POST http.

- http://URL_do_site/messageBuffer/ExtAppMsgInstant
- http://URL_do_site/messageBuffer/ExtAppMsgNormal

Para que a requisição seja processada é necessário um conjunto de variáveis que visam, tanto autenticar o cliente no sistema, como também disponibilizar as informações necessárias para que a mensagens seja processadas pelo sistema.

A seguir é apresentado um quadro resume das variáveis utilizadas:

Tabela 4 - Variáveis dos diferentes POST e GET http

Variável	Descrição	Exemplo
User	username do utilizador no sistema. Utilizado para validação no sistema.	"xxxxx"
UserKey	Chave do utilizador no sistema. O ***userKey*** definido nas configurações limites de envio da conta do utilizador. Utilizado no processo de validação.	"xxxxxxxxx"
Template	Número do template do utilizador, a ser utilizado no processo de validação e envio.	"11"
Key_names	Todos as variáveis separadas pelo Key_separtor. Serão substituídos pelos correspondentes Key_value no protótipo de mensagem existente no template.	Ex1: "#DIA#;#HORA#;" Ex2: "#DIA#~#HORA#~"
Key_separator	Caractere que separa os diferentes Key_names e Key_value	Para ex1: ';' Para ex2: '~'
Key_values	Valores dos Key_names separados pelo Key_separator indicado	Ex1: "03-11-2014;17h30;" Ex2: "03-11-2014~17h30~"
Destination	Telemóvel (com ou sem indicativo) ou email do destinatário da mensagem	"9938309" "+2389938309" "contact@uni-mindelo.edu.cv"
Subject	Assunto da mensagem em caso de envio de email	"teste 001"
Msg	Mensagens a enviar	"Olá! Esta é a minha mensagem 1ª mensagem"

As duas ações (*ExtAppMsgNormal* e *ExtAppMsgInstant*) acedidas através do *controler messageBuffer*, permitem que ambos os métodos GET e POST sejam utilizados. Na resposta a qualquer *request* feito para estes métodos, há um retorno no formato JSON contendo as informações do resultado do pedido.

A seguir é apresentado o quadro resumo das variáveis do JSON de resposta a uma requisição de envio tanto POST como GET:

Tabela 5 - Variáveis do JSON de resposta dos GET e POST

Variável	Descrição	Exemplo
status	Resultado do pedido de envio (OK / erro)	Ex1: "OK" Ex2: "erro"
code	Código de identificação do erro no sistema (000, 001,002,003, 004)	Ex1: "000" Ex2: "002"
message	Mensagem de descrição do erro	Ex1: "Mensagem Recebida" Ex2: "Autenticacao invalida"
msgId	Código único de 20 caracteres que permite acompanhar a mensagem no sistema	Ex1: "kPY$ajJUthb#fsgg56hF"Ex2: "null"

4.2.2.1. Envio Via GET http

O envio de mensagens pela ação *ExtAppMsgInstant* deve conter obrigatoriamente o conteúdo da mensagem a ser enviado passada pela variável Msg.

Sendo que pela ação *ExtAppMsgNormal* não requer mensagem, pois o corpo da mensagem já existe no *template* indicado.

Abaixo é indicado quatro exemplos de URL de envio pelo método POST sendo duas de envio SMS e outras duas de envio de *Email*:

- http://url_do_site_do_sistema/index.php/messageBuffer/ExtAppMsgInstant?User=xxxxx&UserKey=xxxxxx&Template=xx&Destination=99xxx&Msg=xxxxxxxxx

- http://URL_do_site_do_sistema/index.php/messageBuffer/ExtAppMsgNormal?User=xxxxx&UserKey=xxxxxx&Template=xx&Destination=99xxx&Key_names=xxxxxxxx&Key_separator=x&Key_values=xxxxxxx

- http://URL_do_site_do_sistema/index.php/messageBuffer/ExtAppMsgInstant?User=xxxxx&UserKey=xxxxxx&Template=xx&Destination=xxxxx@gmail.com&Msg=xxxxxxxxxxxx&Subject=xxxxxxxx
- http://URL_do_site_do_sistema/index.php/messageBuffer/ExtAppMsgNormal?User=xxxxx&UserKey=xxxxxx&Template=xx&Destination=xxxxx@gmail.com&Subject=xxxxxxKey_names=xxxxxxxx&Key_separator=x&Key_values=xxxxxxx

4.2.2.2. Envio via POST http

O processo de envio de mensagens via POST é, em parte, semelhante ao processo via GET, pois utiliza os mesmos métodos no envio (*ExtAppMsgNormal* e *ExtAppMsgInstant*), sendo que funcione como qualquer outro POST HTTP.

Abaixo segue os dois URLs disponíveis no envio pelo método POST:

- http://URL_do_site_do_sistema/messageBuffer/ExtAppMsgInstant - Os parâmetros *User*, *UserKey*, *Template*, *Msg*, *Destination* são obrigatórios sendo o parâmetro *Subject* opcional e apenas para envios de *email*.

- http://URL_do_site_do_sistema/messageBuffer/ExtAppMsgNormal - Os parâmetros *User*, *UserKey*, *Template*, *Destination*, *Key_names*, *Key_values*, *Key_separator* são obrigatórios sendo o parâmetro *Subject* opcional e apenas para mensagens de *email*.

4.2.2.3. Resposta em formato JSON

Todos os pedidos GET e POST endereçados ao sistema tem como resposta um JSON contendo as descrições do resultado do processamento do pedido no sistema.

Em caso de sucesso, o sistema retorna um JSON com a variável msgId contendo um código que permite ao utilizador verificar posteriormente o estado da mensagem no sistema UMCont@ct.

A seguir é apresentado um exemplo de resposta de um pedido de mensagem feito com sucesso e outro com erro devido a um parâmetro vazio ou por falta de um parâmetro:

Exemplo de pedido de nova mensagem feito com sucesso:

{"***status***":"OK", "***code***":"000", "***message***":"Mensagem Recebida ", "***msgId***":"uPlecd3Td6ht5sbY#kp"}

Exemplo de erro originado por falta de um parâmetro ou por esta estar vasia:

{"***status***":"erro","***code***":"001","***message***":"Parametro vazio","***msgId***":null}

Para além do envio de mensagens por requisição HTTP, o cliente poderá solicitar informações de crédito no sistema e de *status* de mensagem. Para isso é disponibilizado outros dois URLs GET e POST:

- http://URL-DO-SITE/index.php/messageBuffer/Credits - para obter informações sobre utilização e disponibilidade de crédito SMS da sua conta. Parâmetros necessários:
  - username
  - userKey
- http://URL-DO-SITE/index.php/messageBuffer/StatusMSG - para obter informação do estado de uma mensagem no sistema. Parâmetros necessários:
  - username
  - userKey
  - msgId

Parecido o que acontece nas requisições anteriormente descritos, aqui também o sistema retorna um JSON como resposta a cada solicitação.

Sendo que para a ação *Credits* é retornado:

{"status":valor, "code":valor, "SMS":valor, maxSMShour":valor,"maxSMSday":valor, "maxSMSday": valor "maxSMShour":valor,"SMSsendHour":valor, SsendDay":valor}

E para a ação *StatusMSG* é retornado:

{"status":valor, "code":valor, "message":valor}

4.2.3. Envio de Mensagem via Web Service

Para uma melhor integração da UMCont@ct com diferentes plataformas em diferentes linguagens de programação, o sistema disponibiliza uma *Web service*.

Construído em PHP no mesmo *framework* de desenvolvimento da parte administrativa do sistema, e disponibilizado aos clientes através da seguinte URL:

- **http://URL-DO-SITE/index.php/service/quote**

O Web *service* disponibiliza ao clientes um conjunto de funções que permitem enviar SMS e *Email* nos dois modos suportados pelo sistema (*instant*, normal), bem como ter acesso a diversas informações que permitem uma melhor utilização do sistema. Os métodos ou funções disponibilizados são listados abaixo:

- **createSmsInstant()** – método que permite criar um buffer de mensagem SMS de envio instantâneo.
- **createSmsNormal()** – método que permite criar um buffer de mensagem SMS de envio normal.
- **createEmailInstant()** – método ou função que cria um buffer de email de envio instantâneo.
- **createEmailNormal()** – método ou função que cria um buffer de email de envio normal.
- **getCredits()** – método que permite obter informações sobre o estado de crédito do cliente no sistema.
- **getStatusMSG()** – método ou função que permite obter o status de uma mensagem no sistema através do ***msgId*** retornado no momento do envio.

A seguir é apresentado um quadro que permite ter uma melhor perceção dos métodos ou funções disponibilizados, dos parâmetros necessários para a sua chamada, e dos parâmetros da sua resposta.

Tabela 6 - Métodos disponibilizados pelo Web Service do sistema

MÉTODO	PARÂMETROS PARA O PEDIDO	PARÂMETROS DA RESPOSTA
createSmsInstant()	username, userKey, template, destination, message	status, code, message, msgId
createSmsNormal()	username, userKey, template, destination, key_names, key_separator, key_values	
createEmailInstant()	username, userKey, template, destination, subject, message	
createEmailNormal()	username, userKey, template, destination, subject, key_names, key_separator, key_values	
getCredits()	username, userKey	status, code, message, SMS, maxSMSday, maxSMShour, SMSsendHour, SMSsendDay
getStatusMSG()	username, userKey, msgId	status, code, message

Na requisição a qualquer uma das funções do *Web service*, a ordem dos parâmetros acima demostrados é importante para que a requisição seja bem processada pelo sistema.

Em ambas as requisições ao sistema, é retornado um conteúdo SOAP contendo diversas informações que poderão ser utilizadas na tomada de decisão por parte do cliente.

A figura a seguir exemplifica um SOAP contendo o retorno de sucesso da requisição ao método getStatusMSG():

```
<SOAP-ENV:Envelope
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns1="urn:ServiceControllerwsdl"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ns2="http://xml.apache.org/xml-soap"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Body>
        <ns1:getStatusMSGResponse>
            <return xsi:type="ns2:Map">
                <item>
                    <key xsi:type="xsd:string">status</key>
                    <value xsi:type="xsd:string">delivered</value>
                </item>
                <item>
                    <key xsi:type="xsd:string">code</key>
                    <value xsi:type="xsd:string">000</value>
                </item>
                <item>
                    <key xsi:type="xsd:string">message</key>
                    <value xsi:type="xsd:string">Status atual</value>
                </item>
            </return>
        </ns1:getStatusMSGResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 28 - Conteúdo SOAP de retorno de método do Web Service

A imagem que se segue demostra uma função PHP que faz a utilização do Web service para obter informações do *status* de uma mensagem de um cliente.

```
507     public function actionTeste() {
508         $client = new SoapClient('http://sigaum.servehttp.com/um.contact.admin/index.php/service/quote');
509
510         $results = $client->getStatusMSG('teu_username', 'teu_userKey', 'msgId');
511         $string_json = json_encode($results);
512         $json = json_decode($string_json, true);
513
514         echo "      Mensagem da requisicao: " . $json['message'] . "<br>" . "<br>";
515         echo "        Codigo da requisicao: " . $json['code'] . "<br>" . "<br>";
516         echo "        Status da requisicao: " . $json['status'] . "<br>" . "<br>";
517     }
```

Figura 29 - Utilização do Web service do sistema em PHP

Ao executar a função acima, e em caso de sucesso, a saída seria algo como:

Mensagem da requisicao: Status atual Codigo da requisicao: 000 Status da requisicao: delivered

4.3. Códigos de resposta do Sistema

A pensando nos clientes do sistema, numa rápida integração dos seus sistema com a UMC e como forma de facilitar o processo de tratamento das respostas as diversas solicitações feitas ao sistema, foi desenvolvido um conjunto de códigos de respostas.

Assim para cada situação que ocorre numa solicitação é retornada um código de sucesso ou erro, juntamente com uma pequena mensagem de descrição do código.

Na tabela a seguir é apresentado os códigos fornecidos reportados pelo sistema, a mensagem associada a cada código e uma pequena descrição da origem de cada código:

Tabela 7 - Códigos de resposta do sistema

CÓDIGO	MENSAGEM	DESCRIÇÃO
"000"	"Mensagem Recebida" "Creditos do cliente" "Status atual"	Código de sucesso resultante da aceitação do pedido do cliente no sistema. Retornado sempre que há sucesso no pedido do cliente.
"001"	"Parametro vazio"	Código de erro que resulta quando o pedido é feito com um parâmetro vazio ou por falta desta.
"002"	"Autenticacao invalida"	Código de erro resultante quando a autenticação do utilizador no sistema não é bem-sucedido. *username* e o *userKey* inválidos.
"003"	"Template invalido"	Resulta quando o template não existe no sistema, ou se existe não pertence ao utilizador que fez a requisição, ou se pertence, não é do tipo de mensagem que se pretende enviar
"004"	"Erro interno. Por favor tente novamente"	Código retornado quando o motivo é algum problema interno ao sistema, ou o erro é desconhecido
	"MsgId invalido para UMC"	Código de erro quando a msgId que se utiliza na requisição de status de mensagem não está dentro do

"005"		padrão se código de mensagens retornado
"006"	"MsgId desconhecido"	Código de erro originado pela não existência da msgId no sistema

4.4. Log do Servidor do Sistema

Para uma rápida inteiração da situação do servidor Java do sistema, esta produz um log rotativo dividido em 10 ficheiro contendo cada um 10 MB de informações. Para além de registar erros que possam ocorrer no sistema, esta permite acompanhar praticamente milésima à milésima, o estado das diferentes threads que compõe o servidor Java do sistema.

A figura a seguir demonstra o cabeçalho e conteúdo do log:

Date	Time	Thread	Level	Class	Metodo	Line	Message
30/10/2014	16:49:35:923	Thread main	INFO	um_contact_server.Um_contact_server	Metodo main	118	Log session Start time: 26-11-2014 18:50:30
30/10/2014	16:49:35:933	Thread main	ERROR	um_contact_server.Um_contact_server	Metodo main	133	com.mysql.jdbc.exceptions.jdbc4.CommunicationsException: Communications link failure The last packet sent successfully to the server was 0 milliseconds ago. The driver has not received any packets from the server.
30/10/2014	16:49:35:959	Thread Thread-1	DEBUG	aplication.Receiver	Metodo run	280	Receiving ...
30/10/2014	16:49:35:974	Thread Thread-0	DEBUG	aplication.Sender	Metodo run	249	Sending ...
30/10/2014	16:49:35:974	Thread Thread-1	DEBUG	aplication.Receiver	Metodo run	280	Receiving ...
30/10/2014	16:49:35:990	Thread Thread-1	DEBUG	aplication.Receiver	Metodo run	280	Receiving ...
30/10/2014	16:49:35:990	Thread Thread-0	DEBUG	aplication.Sender	Metodo run	249	Sending ...

Figura 30 - Visualização do log do servidor Java

5. CONCLUSÃO

Este trabalho propôs a criação de um serviço de alerta SMS e *Email*, direcionado a clientes particulares, Empresas e outros sistemas. Foi realizado projeto de *software* que fornece ao público-alvo um conjunto de meios para a sua utilização. Baseada nas tecnologia Web para divulgação das sua alerta tanto de *Email* como de SMS.

Na sua implementação foram utilizadas diversas tecnologias, ferramentas e protocolos em que se destaca o Yii framework, PHP OO, Java, MySQL, XML, Web service, SOAP, entre muitos outros. Os obstáculos foram poucos e os que surgiram foram gradualmente sendo ultrapassados, quer pela ajuda de diversas pessoas, quer pelo aprofundar das pesquisas na internet.

Após uma longa e importante fase de teste do protótipo desenvolvido, está se encontra num nível bastante aceitável para se tornar num excelente objeto de negócio.

De referir que os objetivos inicialmente propostos para o sistema não foram alcançados na sua totalidade, uma vez que o módulo no projeto SIGA (Sistema Integrado de Gestão Académica), que iria permitir, principalmente aos estudantes da Universidade do Mindelo, a criação de pedidos de alerta de notas no sistema SIGA, ainda não foi desenvolvido, a atrasos no desenvolvimento de alguns módulo do projeto SIGA, que seria fundamental para a concepção do módulo de alerta.

5.1. Avaliação da Metodologia

De uma forma geral posso dizer que a metodologia de engenharia de *software* utilizada neste projeto, permitiu-me chegar a um resultado final de excelente qualidade, que é o pressuposto quando optamos por essa metodologia.

O excelente nível de controlo sob todas as fases de desenvolvimento do projeto proporcionado por essa metodologia, facilitou, e muito, o trabalho em fase subsequentes. De realçar as poucas dificuldades encontradas na fase de codificação, uma vez que todo o processo nas fases de análise de requisitos e projeto foram bem definidos.

5.2. Avaliação das Ferramentas Utilizadas

A escolha das ferramentas utilizadas na concepção do projeto foi outro aspeto importante na realização do mesmo. As características de cada ferramenta foram bem analisadas previamente como forma de tirar o máximo proveito de cada uma delas, e também como forma de agilizar o todo o processo de desenvolvimento.

Características preponderantes na escolha das ferramentas utilizadas:

- Ser uma ferramenta free – pois por se tratar de um desenvolvimento sem qualquer verba, o custo é um factor decisivo no momento de cada escolha.
- Fácil utilização – com isto pretendia-se o máximo de independência em todo o processo de desenvolvimento do projeto.
- Comunidade de utilizador – em caso de dúvidas e ou problemas a existência de pelo menos uma comunidade ou fórum de esclarecimento sobre o *software* foi outro factor preponderante na escolha.

- Qualidade – em qualquer processo de desenvolvimento, esta característica é fundamental para se chegar a um resultado final de grande nível.

De um modo geral, pode-se dizer que as ferramentas escolhidas permitiram tirar o máximo proveito de cada uma delas, e mais importante, permitiram agilizar todo o processo de desenvolvimento do projeto.

5.3. Conhecimento Adquirido

Durante a realização deste projeto, e com muita dedicaçao, empenho e interacção com o meu orientador Professor Doutor João Dias, muitos foram os conhecimentos adquiridos e outros aprofundados. Conhecimentos que irão servir de uma excelente bagagem para realizarão de futuros projetos .

- Conhecimentos como uso e manipulação de threads Java, que permite um melhor nível de programação quando se trata de programação multitarefas, ou seja, quando queremos que duas ou mais tarefas sejam executadas, simultaneamente, sem um afectar o outro sob uma mesma aplicação.
-
- Técnicas e formas de programação usando a arquitetura MVC (*Model View Controller*), que é a arquitetura padrão de desenvolvimento de aplicativos Web. Tais conhecimentos dessa arquitetura provieram tanto da codificação do servidor em Java como também do framework Yii que foi utilizada na implementação da parte administrativa do projeto.
- Conhecimentos também do PHP mais precisamente da OO (Orientação a Objetos) e suas vantagens na codificação de projetos para Web. Pois a orientação a objetos em PHP está disponível desde a versão PHP5, e tem vindo a aumentar cada vez mais a preferência dessa linguagem para projetos Web.

- Utilização do Yii framework no desenvolvimento de projetos em PHP com manipulação de base de dados.
- Criação manipulação e utilização de Web service em diferentes linguagens de programação.

5.4. Proposta de Trabalho Futuro

Todo e qualquer projeto não terminam após a entrega ao cliente. O acompanhamento e atualizações de sistema deve ser permanente, tanto para melhorar os recursos e serviços disponibilizados, como também para dar novas funcionalidades ao projeto ou *software*. E para tal a concepção de um projeto deve ser o mais genérico possível, permitindo assim futuros atualizações de forma mais fácil possível.

- Como proposta futuro para esse projeto proponho a implementação da tecnologia de MMS, que já está sendo disponibilizado por um dos *gateways* utilizado neste projeto, o Twilio. Esta tecnologia ainda está numa fase de experimentação e disponível apenas no território dos EUA, e assim que disponível no contexto internacional e se os preços praticados assim o justificarem, seria uma mais-valia para este projeto, uma vez que daria uma melhor visão as campanhas publicitaria e de marketing que é uma das ideias de negócio deste projeto.
- Outra ideia é a criação e disponibilização de uma biblioteca ou repositório com exemplos de códigos em diferentes linguagens de programação, permitindo assim fácil acesso e conexão ao sistema por parte dos clientes.
- A pensar nos clientes pretende criar um novo modulo direcionado diretamente a gestão de créditos dos clientes para com o sistema, alertando via SMS e *email* em diferentes situações como escassez de crédito dos clientes para com o serviço, estouro dos limites de envio por hora e por dia, entre outros.
- Validação e confirmação de diferentes ações de clientes por meio de SMS.
- Fornecer aos clientes um serviço de validação de duas vias, por meio de SMS.

- Criação de relatórios numéricos detalhados, sobre a utilização do sistema, como forma de ajudar o administrador do mesmo na tomada de decisões na relação com cliente, ajudando na definição de planos de atuação.

Com estas ideias quero demonstrar que o projeto está numa fase embrionária, e que pretendo ter um processo contínuo de *upgrade* do mesmo, quer com novas funcionalidades como também aprimorando os já existentes.

6. BIBLIOGRAFIAS

MARQUES, Joaquim e SERRÃO, Carlos (2007). *PHP5*. FCA Editora de Informática.

MIGUEL, António (2010). *Gestão de projectos de software*. FCA – Editora de Informática.

LOPES, Filomena. Morais, Maria. Carvalho, Armando (2005). *Desenvolvimento de Sistema de Informação*. FCA Editora de Informática.

PEREIRA, José (1998). *Tecnologia de Base de Dados. 3ª Edição*. FCA Editora de Informática.

SILVA, Alberto (2001). *UML Metodologia e Ferramentas*. Editora Centro Atlântico.

DE SOUZA, André José Martins. *Controlo e Automação de Iluminação Residencial por Telefone*. Monografia de Licenciatura. Centro Universitário de Brasília.

MONTEIRO, Rafael Oliveira (2007). Sistema de Gerenciamento Bibliotecário Utilizando Interface WEB/SMS. Monografia de Licenciatura. Universidade Federal do Rio de Janeiro.

LIBERALQUINO, Diego. *Desenvolvimento de Plataforma de Comunicação GSM/GPRS para Sistemas Embarcados*. Monografia de Bacharel. Universidade de Pernambuco.

SILVA, Bruno Lopes Ribeiro. *Sistema de controle do trio automotivo por meio de SMS*. Monografia de Licenciatura. Centro Universitário de Brasília.

32% dos cabo-verdianos estão "online". Disponível em <**http://www.asemana.publ.cv/spip.php?article80539**>, Consultado em 03/10/2014.

Cabo Verde sobe cinco posições no ranking global da UIT sobre índice de desenvolvimento das TICs. Disponível em <**http://www.anac.cv/index.php?option=comcontent&view=article&id=323%3Acabo-verde-sobe-cinco-posicoes-no-ranking-global-da-uit-sobre-indice-de-desenvolvimento-das-tic&catid=1%3Anoticias-recentes&Itemid=60&lang=pt**> Consultado em 03/10/2014.

dapsms. Disponível em <**http://www.dapsms.com/pt/**> Consultado em 08/04/2014.

SILVA, Márcia. *CRM e Comunicação via SMS*. Disponível em <http://www.phc.pt/portal/programs/ewpview.aspx?codigo=IDIR090826#.U_y0TPnuI1Y> , Consultado em 10/10/2014.

Significado de SMS. Disponível em <**http://www.significados.com.br/sms/**> Consultado em 13/10/2014.

SMS, Comunicação e Marketing. Disponível em <**http://www.mobilepronto.org/comunicacao-marketing-sms.html**> Consultado em 18/06/2014.

Os protocolos de serviço de mensagens (SMTP, POP3 e IMAP4). Disponível em <**http://pt.kioskea.net/contents/282-os-protocolos-de-servico-de-mensagens-smtp-pop3-e-imap4**> Consultado em 16/11/2013.

O Telemóvel na sociedade. Disponível em <**http://www.esds1.pt/site 1514/index.php?option=com_content&view=article&id=485:o-telemovel-na-sociedade &catid=43:sociedade &Itemid=71**> Consultado em 23/10/2014.

SOAP. Disponível em <**http://pt.wikipedia.org/wiki/SOAP**>, Consultado em 27/07/2014.

Engenharia de Software e Linguagens de Programação. Disponível em <**http://unisinos-eslp.blogspot.com/2011/04/rup-x-metodologias-ageis-uma-analise.html**> Consultado em 06/09/2014.

JSON with PHP. Disponível em <http://www.tutorialspoint.com/json/json_php_example.htm> Consultado em 20/07/2014.

What is SOAP? Disponível em <**http://www.tutorialspoint.com/soap/what_is_soap.htm**> Consultado em 20/09/2014.

Yii. Disponível em <**http://pt.wikipedia.org/wiki/Yii**>, Consultado em 20/07/2014.

MySQL. Disponível em <**http://pt.wikipedia.org/wiki/MySQL**> Consultado em 20/07/2014.

NetBeans. Disponível em <**http://pt.wikipedia.org/wiki/NetBeans**>, Consultado em 20/07/2014.

O que é GSM, o que é GPRS? Disponível em <**http://its.wplex.com.br/monitoramento/o-que-e-gsm-o-que-e-gprs/**> Consultado em 20/06/2014.

Web service. Disponível em <**http://pt.wikipedia.org/wiki/Web_service**>, Consultado em 20/10/2014.

VERNET, Oswaldo. *Multithread*. Disponivel em <**http://www.training.com.br/lpmaia/multithread.pdf**>, Consultado em 01/12/2014

7. ANEXO

I.Código das principais classes do servidor Java

[CLASS MAIN - Um_contact_server]

```
package um_contact_server;
import aplication.*;
import dao.MessageBuffer_dao;
import dao.MessageTemplate_dao;
import dao.UserLimits_dao;
import dbconection.ConectionDB;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.sql.Connection;
import java.sql.SQLException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.List;
import java.util.Properties;
import model.MessageBuffer;
import org.apache.commons.configuration.ConfigurationException;
import org.apache.commons.configuration.PropertiesConfiguration;
import org.apache.log4j.*;

/**
 *
 * @author Ivanir Gomes
 */
public final class Um_contact_server extends Thread {

   private static final Logger logger = Logger.getLogger("aplication.Um_contact_server");
   final static String propFileName = "DBserveConfig.txt";

   private static void createConfig() throws IOException, ConfigurationException {
      if (!new File(propFileName).exists()) {
         new File(propFileName).createNewFile();
      }
      PropertiesConfiguration prop = new PropertiesConfiguration(propFileName);
      prop.setProperty("server", "127.0.0.1");
      prop.setProperty("dataBase", "umcontact");
      prop.setProperty("user", "root");
      prop.setProperty("password", "");
      prop.setProperty("log4j.rootLogger", "TRACE,A1,R");
      prop.setProperty("log4j.appender.A1", "org.apache.log4j.ConsoleAppender");
      prop.setProperty("log4j.appender.A1.layout", "org.apache.log4j.PatternLayout");
      prop.setProperty("log4j.appender.A1.layout.ConversionPattern", "%-d{yyyy-MM-dd HH:mm:ss,SSS} [%c]-[%p]
%m%n");
      prop.setProperty("log4j.appender.R", "org.apache.log4j.RollingFileAppender");
      prop.setProperty("log4j.appender.R.File", "log/umcontactlog.html");
      prop.setProperty("log4j.appender.R.MaxFileSize", "10MB");
      prop.setProperty("log4j.appender.R.MaxBackupIndex", "10");
      prop.setProperty("log4j.appender.R.layout", "aplication.utils.UMContactServerHTMLLayout");
      prop.save();
   }

   //CRIA A CONFIGURACAO DE BASE DE DADOS A PARTIR DA LEITURA DAS CONFIGURACOES DO
FICHEIRO
```

```
private static ConectionDB lerConfigDB() throws IOException, ConfigurationException {
    FileInputStream inputStream = new FileInputStream(propFileName);
    Properties prop = new Properties();
    if (inputStream == null) {
        createConfig();
        prop.load(inputStream);
    } else {
        prop.load(inputStream);
    }
    // get the property value
    String servidor = prop.getProperty("server");
    String baseDados = prop.getProperty("dataBase");
    String user = prop.getProperty("user");
    String pass = prop.getProperty("password");
    return new ConectionDB(servidor, baseDados, user, pass);
}

private static void resetStatusBuffer(Connection conexao) {
    try {
        List<MessageBuffer> lista = new MessageBuffer_dao(conexao).listaMBufferStatus("PR");
        if (!lista.isEmpty()) {
            for (MessageBuffer buffer : lista) {
                buffer.setStatus("RC");
                new MessageBuffer_dao(conexao).actualizarBuffer(buffer);
            }
        }
        List<MessageBuffer> lista2 = new MessageBuffer_dao(conexao).listaMBufferStatus("FL");
        if (!lista.isEmpty()) {
            for (MessageBuffer buffer2 : lista2) {
                buffer2.setStatus("RC");
                new MessageBuffer_dao(conexao).actualizarBuffer(buffer2);
            }
        }
    } catch (SQLException ex) {
        logger.log(Level.ERROR, ex);
    }
}

private static void checkNotification() {
    try {
        if (new MessageTemplate_dao(lerConfigDB().getConnection()).systemTemplateEmail() == null) {
            new MessageTemplate_dao(lerConfigDB().getConnection()).createTemplEMAILsystem();
        }
        if (new MessageTemplate_dao(lerConfigDB().getConnection()).systemTemplateSMS() == null) {
            new MessageTemplate_dao(lerConfigDB().getConnection()).createTemplSMSsystem();
        }
    } catch (IOException | ConfigurationException | SQLException ex) {
        logger.log(Level.ERROR, ex);
    }
}

public static void main(String args[]) {
    try {
        if (!new File(propFileName).exists()) {
            if (new File(propFileName).createNewFile()) {
                createConfig();
            }
        }
        PropertyConfigurator.configure(propFileName);
```

```
        SimpleDateFormat format = new SimpleDateFormat("dd-MM-yyyy HH:mm:ss");
        logger.log(Level.INFO, "Log session Start time: " + format.format(new Date()));
        if (new UserLimits_dao(lerConfigDB().getConnection()).userIdOfUserSystem() != 0) {
            checkNotification();
            ConectionDB conectionDB = lerConfigDB();

            resetStatusBuffer(conectionDB.getConnection());
            Thread thread = new Thread(new UMmonitor(conectionDB));
            thread.start();
        } else {
            System.err.println("Não foi detectado o user utilizado pelo sistema.\nPor favor cria o user \"UMCsystem\" utilizado nas notificaçoes do sistema!");
            logger.log(Level.FATAL, "Não foi detectado o user utilizado pelo sistema.\nPor favor cria o user \"UMCsystem\" utilizado nas notificaçoes do sistema!");
            System.exit(0);
        }
    } catch (IOException | ConfigurationException ex) {
        logger.log(Level.FATAL, ex);
    } catch (SQLException ex) {
        logger.log(Level.ERROR, ex);
        System.out.println(ex);
    }
  }
}
```

[FIM DA CLASS MAIN]

[CLASS RECEIVER]

```
import aplication.utils.UMThreadStatus;
import aplication.utils.UMContactSistemStatusEnum;
import model.*;
import dao.*;
import java.sql.Connection;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
import org.apache.log4j.*;

/**
 *
 * @author Ivanir Gomes
 */
public class Receiver implements Runnable {

  private static final Logger logger = Logger.getLogger(Receiver.class);
  private Connection conexao = null;
  UMThreadStatus status = null;
  UMmonitor monitor = null;
  ServerControl control = null;

  private boolean firstDesconectBD = true;

  public Receiver(UMmonitor mon, Connection con) {
    this.conexao = con;
    this.monitor = mon;
    this.status = monitor.getStatus();
    this.status.setReceiving(true);
    control = new ServerControl_dao(conexao).myControl();
    if (control.getStatus().equalsIgnoreCase("KILL")) {
```

```
            logger.log(Level.ERROR, "System killing by UMC " + control.getUser());
            System.exit(0);
        }
    }

    /*METODO QUE FAZ A VALDACAO DA MESSAGEM RECEBIDA DO UTILIZADOR. VALIDACAO DE LIMITES DE ENVIO, VALIDACAO DA ORIGEM DA MSG, VALIDACAO DE PARAMETROS E SEU TEMPLATE
     RECEBE TODAS AS MSGNO STATUS RC, E SE PASSAR PELA VALIDACAO O STATUS PASSA PARA SC SE FOR UMA MSG DO TIPO INSTANANEO (SNAPSHOT EM INGLES), OU PARA RD (READY) SE FOR UMA MSG DO TIPO AGENDADO*/
    private void validarBuffer(MessageBuffer buffer) {

        String erroMSG = "";
        if (buffer != null && buffer.getStatus().equalsIgnoreCase("RC")) {

            if (buffer != null && buffer.getTemplate_id() != 0 && buffer.getDestination() == null) {
                erroMSG += "Mensagem sem destinatario.\n";
            }
            if (buffer.getTemplate_id() == 0) {
                erroMSG += "Mensagem sem template_id.\n";
            }
            //pegar o template correspondente
            try {
                MessageTemplate template = new MessageTemplate_dao(conexao).templateOfBuffer(buffer.getTemplate_id());

                if (template != null && !template.isActive()) {
                    erroMSG += "Template desactivado.\n";
                }

                if (buffer.getTemplate_id() != 0 && template == null) {
                    erroMSG += "Template correspondente desconhecido.\n";
                }
                if (buffer.getTemplate_id() != 0 && template.getMsg_type() == null) {
                    erroMSG += "Tipo de Menssagem null no template de buffer (Informe o tipo SMS ou EMAIL.\n";
                }
                if (buffer.getTemplate_id() != 0 && template.getMsg_type() != null && !template.getMsg_type().equalsIgnoreCase("SMS") && !template.getMsg_type().equalsIgnoreCase("EMAIL")) {
                    erroMSG += "Tipo de Menssagem desconhecido pelo sistema.\n";
                }
                if (buffer.getTemplate_id() != 0 && template.getTemplate_type() == null) {
                    erroMSG += "Null em \"TEMPLATE_TYPE\" no template deste Buffer.\n";
                }
                if (buffer.getTemplate_id() != 0 && template.getTemplate_type() != null && !template.getTemplate_type().equalsIgnoreCase("normal") && !template.getTemplate_type().equalsIgnoreCase("instant")) {
                    erroMSG += "Tipo de template invalido para este buffer(Informe se é \"normal ou instant\".\n";
                }
                if (buffer.getTemplate_id() != 0 && template.getTemplate_type() != null && template.getTemplate_type().equalsIgnoreCase("normal") && buffer.getBuilt_msg() == null && (buffer.getParameter_names() == null || buffer.getParameter_separator() == null || buffer.getParameter_value() == null)) {
                    erroMSG += "Campos de Parameter vasio no buffer.\n";
                }

                if (buffer.getTemplate_id() != 0 && template.getOwner_user() == 0) {
                    erroMSG += "Owner_User invalido no template do buffer.\n";
                }

                //VALIDACAO DE PARAMETROS
                if (template != null && template.getTemplate_type().equalsIgnoreCase("normal") && buffer.getBuilt_msg() == null
```

```
                && buffer.getParameter_names() != null && buffer.getParameter_separator() != null &&
buffer.getParameter_value() != null) {
                    //nº de parametro name
                    int par_name = 0;
                    //nº de parameros value
                    int par_val = 0;
                    //contar o numero de parameter_name
                    for (int i = 0; i < buffer.getParameter_names().length(); i++) {
                        if (String.valueOf(buffer.getParameter_names().charAt(i)).equals(buffer.getParameter_separator())) {
                            par_name += 1;
                        }
                    }
                    //contar o numero de parameter_value
                    for (int i = 0; i < buffer.getParameter_value().length(); i++) {
                        if (String.valueOf(buffer.getParameter_value().charAt(i)).equals(buffer.getParameter_separator())) {
                            par_val += 1;
                        }
                    }
                    if ((par_name == 0 || par_val == 0) || (par_name != par_val)) {
                        erroMSG += "Erro nos parametros.\n";
                    }
                }

                if (buffer.getTemplate_id() != 0 && template.getTemplate_type() != null &&
template.getTemplate_type().equalsIgnoreCase("instant") && (buffer.getBuilt_msg() == null ||
buffer.getBuilt_msg().isEmpty())) {
                    erroMSG += "Messagem instantaneo sem Built_MSG.\n";
                }

                //validacao do user
                UserLimits utilizador = new UserLimits_dao(conexao).UserAccount(template.getOwner_user());
                //verificar se o utilizador de origem da mensagem existe
                if (buffer.getTemplate_id() != 0 && utilizador == null) {
                    erroMSG += "Utilizador inexistente.\n";
                }

                if (buffer.getTemplate_id() != 0 && template.getMsg_type().equalsIgnoreCase("SMS") && template.getIndicativo()
== null) {
                    erroMSG += "Template de SMS sem indicative.\n";
                }

                /*VERIFICADO TODOS OS POSSIVEIS ERROS, VAMOS DAR UM FEEDBACK E OU ACUTALIZAR
STATUS*/
                if (!erroMSG.isEmpty()) {
                    buffer.setStatus("FL");
                    buffer.setError_code(null);
                    buffer.setError_msg(erroMSG);
                    actualizarBuffer(buffer);
                } else if (template.getTemplate_type() != null && template.getTemplate_type().equalsIgnoreCase("instant")) {
                    buffer.setMsg_type(template.getMsg_type());
                    buffer.setSource_System(template.getOwner_system());
                    buffer.setControl_Caracteres(template.isControl_caracteres());
                    if (buffer.getUser_ID() == 0) {
                        buffer.setUser_ID(template.getOwner_user());
                    }

                    if (template.getMsg_type().equalsIgnoreCase("SMS")) {
                        buffer.setDestination(buffer.getDestination().replaceAll(" ", ""));
```

```
                if (!buffer.getDestination().startsWith("+") && !buffer.getDestination().startsWith("00") &&
!buffer.getDestination().startsWith(template.getIndicativo())) {
                    buffer.setDestination(template.getIndicativo() + buffer.getDestination());
                }
            }
            buffer.setStatus("SC");
            actualizarBuffer(buffer);
        } else if (template.getTemplate_type() != null && template.getTemplate_type().equalsIgnoreCase("normal")) {
            buffer.setMsg_type(template.getMsg_type());
            buffer.setSource_System(template.getOwner_system());
            buffer.setControl_Caracteres(template.isControl_caracteres());
            if (buffer.getUser_ID() == 0) {
                buffer.setUser_ID(template.getOwner_user());
            }

            if (template.getMsg_type().equalsIgnoreCase("SMS")) {
                buffer.setDestination(buffer.getDestination().trim());
                if (!buffer.getDestination().startsWith("+") && !buffer.getDestination().startsWith("00") &&
!buffer.getDestination().startsWith(template.getIndicativo())) {
                    buffer.setDestination(template.getIndicativo() + buffer.getDestination());
                }
            }
            buffer.setStatus("RD");
            actualizarBuffer(buffer);
        } else {
            buffer.setStatus("FL");
            buffer.setError_code(null);
            buffer.setError_msg("Tipo de template desconhecido para este Buffer");
            actualizarBuffer(buffer);
        }
    } catch (SQLException ex) {
        logger.log(Level.ERROR, ex);
    }
  }
}

  /*CRIA A MENSAGEM A SER ENVIADA, ATRAVES DO SEU TEMPLATE E OS PARAMETROS VALUE FORNECIDO PELO USER
   OU SEJA FAZ A SUBSTITUICAO DOS PARAMETROS NAMES DO TEMPLATE PELOS RESPECTIVOS PARAMETROS VALUE*/
  private void builderBuffer(MessageBuffer buffer) {
    if (buffer != null) {
      try {
        if (buffer.getRetry_number() < 1 && buffer.getBuilt_msg() == null || buffer.getBuilt_msg().equals("")) {
          //COLOCAR O BUFFER EM STATUS DE PROCESSAMENTO
          buffer.setStatus("PR");
          new MessageBuffer_dao(conexao).actualizarBuffer(buffer);
          ///////translate template//////
          MessageTemplate template = new MessageTemplate_dao(conexao).templateOfBuffer(buffer.getTemplate_id());
          String message = template.getMsg();
          String par_names = buffer.getParameter_names();
          String par_value = buffer.getParameter_value();
          String Par_separ = buffer.getParameter_separator();

          if (par_names != null && par_value != null && Par_separ != null) {
            int x = 0, x2 = 0, y = 0, y2 = 0;

            for (x = x2; x < par_names.length(); x++) {
              if (String.valueOf(par_names.charAt(x)).equals(buffer.getParameter_separator())) {
```

```
                    String pname = par_names.substring(x2, x);
                    x2 = x + 1;
                    for (y = y2; y < par_value.length(); y++) {
                        if (String.valueOf(par_value.charAt(y)).equals(buffer.getParameter_separator())) {
                            String pvalue = par_value.substring(y2, y);
                            y2 = y + 1;
                            message = message.replaceAll(pname, pvalue);
                            break;
                        }
                    }
                }
            }
        }
        buffer.setBuilt_msg(message);
        try {
            //COLOCAR O BUFFER EM STATUS DE AGENDADO E SET A HORA E DATE DE ENVIO
            buffer.setStatus("SC");
            actualizarBuffer(buffer);
        } catch (SQLException ex) {
            logger.log(Level.ERROR, ex);
        }
    } else {
        //COLOCAR O BUFFER EM STATUS DE AGENDADO E SET A HORA E DATE DE ENVIO
        buffer.setStatus("SC");
        actualizarBuffer(buffer);
    }
} catch (SQLException ex) {
    logger.log(Level.ERROR, ex);
}
}
}

private void validate() {
    List<MessageBuffer> lista = new ArrayList<>();
    try {
        lista = new MessageBuffer_dao(conexao).listaMBufferStatus("RC");
        if (lista != null && !lista.isEmpty()) {
            for (MessageBuffer buffer : lista) {
                validarBuffer(buffer);
            }
        }
    } catch (SQLException ex) {
        logger.log(Level.ERROR, ex);
    }
}

private void msgBuilder() {
    List<MessageBuffer> lista = null;
    try {
        lista = new MessageBuffer_dao(conexao).listaMBufferStatus("RD");
        if (lista != null && !lista.isEmpty()) {

            for (MessageBuffer buffer : lista) {
                builderBuffer(buffer);
            }
        }
    } catch (SQLException ex) {
        logger.log(Level.ERROR, ex);
    }
```

```
    }

    //metodo para actualizar status de um buffer
    private void actualizarBuffer(MessageBuffer buffer) throws SQLException {
        new MessageBuffer_dao(conexao).actualizarBuffer(buffer);
    }

    @Override
    public void run() {
        while (status.isReceiveing()) {
            if (control.getStatus().equalsIgnoreCase("RUNNING")) {
                try {
                    //verifica se existe conexao com a base de dados ou se a conexao esta fechada
                    if (conexao == null || conexao.isClosed()) {
                        new Thread(this).sleep(5000);//tenta recuperar a conecao a cada 5 segundos
                        conexao = monitor.getConexao();
                        logger.log(Level.INFO, "conexao com DB recuperada no sender");
                        firstDesconectBD = true;
                    } else {

                        logger.log(Level.DEBUG, "Receiving ...");
                        validate();
                        msgBuilder();
                        if (this.status.getSisOverallStatus() == UMContactSistemStatusEnum.PAUSED) {

                            System.out.println("Thread receiver on paused...");
                            try {
                                logger.log(Level.INFO, "Thread receiver paused...");
                                new Thread(this).sleep(5000);
                                System.out.println("Thread receiver retomando execucao...");
                                logger.log(Level.INFO, "Thread receiver retomando execucao...");

                            } catch (InterruptedException ex) {
                                logger.log(Level.ERROR, ex);
                                status.stopReceiving();
                            }
                        }
                    }
                } catch (SQLException ex) {
                    //registra no log apenas a primeira falha de conexao com bd, sendo as autras tentativas ignoradas
                    if (firstDesconectBD) {
                        firstDesconectBD = false;
                        logger.log(Level.ERROR, ex);
                    }
                    System.out.println(ex.getMessage());
                } catch (InterruptedException ex) {
                    logger.log(Level.ERROR, ex);
                }
            }
            control = new ServerControl_dao(conexao).myControl();
            if (control.getStatus().equalsIgnoreCase("KILL")) {
                logger.log(Level.ERROR, "System killing by UMC " + control.getUser());
                System.exit(0);
            }
        }
        logger.log(Level.INFO, "Thread receiver vai terminar...");
    }

}
```

[FIM CLASS RECEIVER]

[CLASS SENDER

```
package aplication;
import aplication.utils.UMThreadStatus;
import aplication.http.HttpSender;
import aplication.smtp.SmtpSender;
import aplication.utils.GatewayResponse;
import aplication.utils.UMContactSistemStatusEnum;
import dao.*;
import java.io.IOException;
import java.sql.Connection;
import java.sql.SQLException;
import java.util.List;
import model.*;
import org.apache.commons.mail.EmailException;
import org.apache.log4j.*;

/**
 *
 * @author Ivanir Gomes
 */
public class Sender implements Runnable {

    private static final Logger logger = Logger.getLogger(Sender.class);
    private Connection conexao = null;
    UMmonitor monitor = null;
    UMThreadStatus status = null;
    List<MessageBuffer> listaMBufferToSend = null;
    MessageBuffer bufferSend;
    ConfiguracaoSMTP SMTP = null;
    GatewaySMS GTWAY = null;
    private boolean firstDesconectBD = true;
    ServerControl control = null;
    private boolean notificarInSuspended = true;
    private final String EstadoAnterior;
    private boolean inicio = true;

    public Sender(UMmonitor mon, Connection con) {
        this.monitor = mon;
        this.status = monitor.getStatus();
        this.conexao = con;
        this.status.setSending(true);
        control = new ServerControl_dao(conexao).myControl();
        if (control.getStatus().equalsIgnoreCase("KILL")) {
            logger.log(Level.ERROR, "System killing by UMC " + control.getUser());
            System.exit(0);
        }
        EstadoAnterior = control.getStatus();
    }

    private void iniciarVariaveis() {
        try {
            if (conexao != null && !conexao.isClosed()) {
                this.SMTP = new ConfiguracaoSMTP_dao(conexao).smtp_activo();
                this.GTWAY = new GatewaySMS_dao(conexao).gatewayActivo();
            }
        } catch (SQLException ex) {
            SMTP = null;
```

```
            GTWAY = null;
            logger.log(Level.ERROR, "erro no metodo iniciarVariaveis: " + ex);
        }
    }

    //retorna a lista d msg no status agendado "SC", prontos a serem enviados
    private void getMsgToBeSent() {
        try {
            this.listaMBufferToSend = new MessageBuffer_dao(conexao).listaMBufferToSend();
        } catch (SQLException ex) {
            logger.log(Level.ERROR, "erro no metodo getMsgToBeSent: " + ex);
        }
    }

    //faz o control e distribuicao para das msg para os respectivos metidos de envio
    private void send() {
        if (listaMBufferToSend != null && !listaMBufferToSend.isEmpty()) {
            for (MessageBuffer buffer : listaMBufferToSend) {
                if (buffer != null && buffer.getDestination() != null) {
                    //verifica se é um email e valida os limites de envio
                    if (buffer.getMsg_type().equalsIgnoreCase("EMAIL") && SMTP != null && validLimitSend(buffer)) {
                        if (buffer.getDestination().contains("@") && !buffer.getDestination().startsWith("@") &&
!buffer.getDestination().endsWith("@") && !buffer.getDestination().endsWith(".")
                                && buffer.getDestination().substring(buffer.getDestination().indexOf("@"),
buffer.getDestination().length()).contains(".")) {
                            try {
                                buffer.setRetry_number(buffer.getRetry_number() + 1);
                                buffer.setStatus("PR");
                                buffer.setError_code(null);
                                buffer.setError_msg(null);
                                actualizarStatus(buffer);
                                //VERIFICA SE TRATA DA MSG DE EMAIL DE NOTIFICACAO QUE HOUVE UM STOPED
                                if (control.getStatus().equalsIgnoreCase("SUSPENDED") && buffer.getSubject().equals("Notificação de
Status UMCserver")) {
                                    this.notificarInSuspended = false;
                                }
                                new SmtpSender().enviarEmail(SMTP, buffer.getDestination(), buffer.getSubject(),
buffer.getBuilt_msg());
                                buffer.setStatus("OK");
                                buffer.setError_code(null);
                                buffer.setError_msg(null);
                                actualizarStatus(buffer);
                            } catch (EmailException ex) {
                                if (ex.toString().contains("Sending the email to the following server failed")) {
                                    buffer.setStatus("SC");
                                    buffer.setError_code(null);
                                    buffer.setError_msg(ex.toString());
                                    actualizarStatus(buffer);
                                } else {
                                    buffer.setStatus("FL");
                                    buffer.setError_code(null);
                                    buffer.setError_msg(ex.toString());
                                    actualizarStatus(buffer);
                                }
                                logger.log(Level.ERROR, "erro no metodo enviarEmail: " + ex);
                            }
                        } else {
                            buffer.setStatus("FL");
                            buffer.setError_code(null);
```

```
                    buffer.setError_msg("E-mail de destinatario invalido!");
                    actualizarStatus(buffer);
                }
            }
            //verifica se é um sms e valida os limites de envio
            if (buffer.getMsg_type().equalsIgnoreCase("SMS") && GTWAY != null && validLimitSend(buffer)) {
                //retira os espaços entre os digitos
                buffer.setDestination(buffer.getDestination().replace(" ", ""));
                //verifica se contem apenas numeros
                if (buffer.getDestination().matches("[0-9+]*")) {
                    try {
                        buffer.setStatus("PR");
                        buffer.setRetry_number((buffer.getRetry_number() + 1));
                        actualizarStatus(buffer);
                        //VERIFICA SE TRATA DA MSG DE NOTIFICACAO DE STOPED
                        if (control.getStatus().equalsIgnoreCase("SUSPENDED") &&
buffer.getDestination().equals(control.getTelemovel())
                                && buffer.getBuilt_msg().contains("Alteracao de Status da UMCserver em") &&
buffer.getOriginator().equals("SERVER UMC")
                                && buffer.getSubject().equals("Notificacao de Status UMCserver")) {
                            this.notificarInSuspended = false;
                        }
                        /*INDEPENDENTEMENTE DE GATEWAY SMS UTILIZADO, O RETORNO SERA SEMPRE UM
OBJECTO GatewayResponse
                         COM TODAS AS INFORMAÇOES NECESSARIA PARA CONTINUACAO DO
PROCESSAMENTO*/
                        GatewayResponse response = new HttpSender().sendSMS(GTWAY, buffer);
                        if (response != null) {
                            new MessageBuffer_dao(conexao).messageSended(response);
                        }
                    } catch (IOException ex) {
                        buffer.setStatus("SC");
                        buffer.setError_code(null);
                        buffer.setError_msg(ex.toString());
                        actualizarStatus(buffer);
                        logger.log(Level.ERROR, "Erro ao enviar msg: " + ex);
                        System.err.println(ex);
                    } catch (SQLException ex) {
                        logger.log(Level.ERROR, "Erro ao processar resultado do envio: " + ex);
                        System.err.println(ex);
                    }
                } else {
                    buffer.setStatus("FL");
                    buffer.setError_code(null);
                    buffer.setError_msg("Destinatario de SMS invalido !");
                    actualizarStatus(buffer);
                }
            }
        }
    }
}

//validar os limite de envio de SMS e EMAIL do utilizador
private boolean validLimitSend(MessageBuffer buffer) {
    if (buffer != null) {
        try {
            if (buffer.getTemplate_id() != 0) {
                UserLimits utilizador = new UserLimits_dao(conexao).limitsOfUserId(buffer.getUser_ID());
```

```
            if (buffer.getMsg_type().equalsIgnoreCase("SMS")) {
                if (utilizador != null && utilizador.getMax_sms() > 0) {
                    if (utilizador.getMax_sms_day() > smsInDay(utilizador.getUser_ID())) {

                        if (utilizador.getMax_sms_hour() <= smsInHour(utilizador.getUser_ID())) {
                            System.out.println("limite de sms por hora esgotado!");
                            return false;
                        } else {
                            return true;
                        }
                    } else {
                        System.out.println("limite de sms por dia esgostado!");
                        return false;
                    }
                } else {
                    return false;
                }
            } else if (buffer.getMsg_type().equalsIgnoreCase("EMAIL")) {
                if (utilizador.getMax_mail_day() > mailInDay(utilizador.getUser_ID())) {

                    if (utilizador.getMax_mail_hour() == mailInHour(utilizador.getUser_ID())) {
                        System.out.println("limite de email por hora esgotado!");
                        return false;
                    } else {
                        return true;
                    }
                } else {
                    System.out.println("limite de email por dia esgostado!");
                    return false;
                }
            }
        } else {
            return true;
        }
    } catch (SQLException ex) {
        logger.log(Level.ERROR, "erro no metodo validLimitSend: " + ex);
    }
  }
  return false;
}

//metodo para actualizar status de um buffer
private void actualizarStatus(MessageBuffer buffer) {
  try {
      new MessageBuffer_dao(conexao).actualizarBuffer(buffer);
  } catch (SQLException ex) {
      logger.log(Level.ERROR, ex);
  }
}

private int smsInDay(int user_id) throws SQLException {
  return new MessageBuffer_h_dao(conexao).countSMSinDay(user_id);
}

private int smsInHour(int user_id) throws SQLException {
  return new MessageBuffer_h_dao(conexao).countSMSinHour(user_id);
}

private int mailInDay(int user_id) throws SQLException {
```

```
        return new MessageBuffer_h_dao(conexao).countMAILinDay(user_id);
    }

    private int mailInHour(int user_id) throws SQLException {
        return new MessageBuffer_h_dao(conexao).countMAILinHour(user_id);
    }

    @Override
    public void run() {
        while (status.isSending()) {
            if (control.getStatus().equalsIgnoreCase("RUNNING")
                    || (control.getStatus().equalsIgnoreCase("SUSPENDED") && !inicio && this.notificarInSuspended &&
(control.isNotificarEmail() || control.isNotificarSMS()))) {

                inicio = false;
                try {
                    if (conexao == null || conexao.isClosed()) {
                        new Thread(this).sleep(5000);//tenta recuperar a conecao a cada 5 segundos
                        conexao = monitor.getConexao();
                        logger.log(Level.INFO, "conexao com DB recuperada no sender");
                        firstDesconectBD = true;
                    } else {
                        logger.log(Level.DEBUG, "Sending ...");
                        //actualizar as variaveis SMTP E GTWAY
                        iniciarVariaveis();
                        //pega as mensagem prontos para envio
                        getMsgToBeSent();
                        //envia as mensagens
                        send();
                        if (status.getSisOverallStatus() == UMContactSistemStatusEnum.PAUSED) {
                            try {
                                logger.log(Level.INFO, "Thread sender on pause...");
                                new Thread(this).sleep(status.getMilisecsOnPuase());
                                System.out.println("Thread send retomando execucao...");
                                logger.log(Level.INFO, "Thread send retomando execucao...");

                            } catch (InterruptedException ex) {
                                logger.log(Level.ERROR, ex);
                            }
                        }
                    }
                } catch (SQLException ex) {
                    //registra no log apenas a primeira falha de conexao com bd, sendo as autras tentativas ignoradas
                    if (firstDesconectBD) {
                        firstDesconectBD = false;
                        logger.log(Level.ERROR, ex);
                    }
                } catch (InterruptedException ex) {
                    logger.log(Level.ERROR, ex);
                }
            }
            control = new ServerControl_dao(conexao).myControl();
            if (control.getStatus().equalsIgnoreCase("KILL")) {
                logger.log(Level.ERROR, "System killing by UMC " + control.getUser());
                System.exit(0);
            }
            if (!control.getStatus().equalsIgnoreCase(EstadoAnterior)) {
                notificarInSuspended = true;
            }
```

```
        }

        logger.log(Level.INFO, "Thread sender vai terminar...");
    }

}
```

[FIM CLASS SENDER]
[CLASS MONITOR]

```
package aplication;

import aplication.utils.UMThreadStatus;
import aplication.utils.UMContactSistemStatusEnum;
import dao.MessageBuffer_dao;
import dao.MessageTemplate_dao;
import dao.ServerControl_dao;
import dbconection.ConectionDB;
import java.sql.Connection;
import java.sql.SQLException;
import java.util.List;
import model.MessageBuffer;
import model.MessageTemplate;
import model.ServerControl;
import org.apache.log4j.*;

/**
 *
 * @author Ivanir Gomes
 */
public class UMmonitor implements Runnable {

    //logger para a class
    private static final Logger logger = Logger.getLogger(UMmonitor.class);
    private Connection conexao;
    private ConectionDB conectionDB;
    private static final UMThreadStatus status = new UMThreadStatus();
    Receiver receiver;
    Sender sender;
    Thread threadSender;
    Thread threadReceiver;
    final int ThreadTimeOut = 20000;//20 segundos
    final int ThreadPauseTimeOut = 6000;//6 segundo
    ServerControl control = null;
    private String EstadoAnterior;

    public UMmonitor(ConectionDB conDB) {

        try {

            this.conectionDB = conDB;
            this.conexao = conectionDB.getConnection();

            receiver = new Receiver(this, conexao);
            sender = new Sender(this, conexao);
            //thread de envio
            threadSender = new Thread(sender);
            // thread receiver
            threadReceiver = new Thread(receiver);
            // actualiza o estado actual do Servidor (PLAY, STOP, RESTART)
```

```
            control = new ServerControl_dao(conexao).myControl();
            if (control.getStatus().equalsIgnoreCase("KILL")) {
                logger.log(Level.ERROR, "System killing by UMC " + control.getUser());
                System.exit(0);
            }
            EstadoAnterior = control.getStatus();
        } catch (SQLException ex) {
            logger.log(Level.ERROR, ex);
        }
    }

    public static UMThreadStatus getStatus() {
        return status;
    }

    private void startThreads() {

        if (threadReceiver != null && !threadReceiver.isAlive()) {
            logger.log(Level.INFO, "Receiving starting...");
            threadReceiver.start();
            logger.log(Level.INFO, "Thread Receiver started.");
        }

        if (threadReceiver == null) {
            receiver = new Receiver(this, conexao);
            threadReceiver = new Thread(receiver);
            threadReceiver.start();
            logger.log(Level.INFO, "Thread Receiver restarted.");
        }

        if (threadSender != null && !threadSender.isAlive()) {

            logger.log(Level.INFO, "Sending starting..");
            threadSender.start();
            logger.log(Level.INFO, "Thread sender started.");
        }

        if (threadSender == null) {
            sender = new Sender(this, conexao);
            threadSender = new Thread(sender);
            threadSender.start();
            logger.log(Level.INFO, "Thread Sender restarted.");
        }

        this.status.setSisOverallStatus(UMContactSistemStatusEnum.RUNNING);  //indicar que o sistama esta a correr
        logger.log(Level.INFO, "Sistem Status: " + UMContactSistemStatusEnum.RUNNING.toString());
    }

    private void stopThreads() {

        try {
            if (status != null) {
                status.stopSending();
                status.stopReceiving();
                //this.status.setSisOverallStatus(UMContactSistemStatusEnum.STOPPED);
            }
            if (threadSender != null) {
                logger.log(Level.INFO, "Waiting for sender to died...");
                threadSender.join(ThreadTimeOut);
```

```
            threadSender = null;
            logger.log(Level.INFO, "sender is dead");

        }
        if (threadReceiver != null) {
            logger.log(Level.INFO, "Waiting for receiver to died...");
            threadReceiver.join(ThreadTimeOut);
            threadReceiver = null;
            logger.log(Level.INFO, "receiver is dead");
        }

    } catch (InterruptedException ex) {
        logger.log(Level.ERROR, ex);
    }
}

private void restartThreads() {
    stopThreads();
    logger.log(Level.INFO, "Sender AND Receiver stopped for restart...");
    resetStatusBuffer();
    startThreads();
    logger.log(Level.INFO, "Sender AND Receiver started in restart...");
}

private void monitor_Threads() {
    if (this.status.getSisOverallStatus() == UMContactSistemStatusEnum.RUNNING
            && (!threadReceiver.isAlive() || !threadSender.isAlive() || status.isHasIntdeterminedError())) {
        restartThreads();
        return;
    }
    if (this.status.getSisOverallStatus() == UMContactSistemStatusEnum.PAUSED) {
        this.status.setMilisecsOnPuase(ThreadPauseTimeOut);
    }
    logger.log(Level.TRACE, "Receiver State: " + threadReceiver.getState().toString());
    logger.log(Level.TRACE, "Sender State: " + threadSender.getState().toString());
}

private void arquivarBuffer() {
    try {
        new MessageBuffer_dao(conexao).arquivarMBufferSended();
    } catch (SQLException ex) {
        logger.log(Level.ERROR, ex);
    }
}

private void resetStatusBuffer() {
    try {
        List<MessageBuffer> lista = new MessageBuffer_dao(conexao).listaMBufferStatus("PR");
        if (!lista.isEmpty()) {
            for (MessageBuffer buffer : lista) {
                buffer.setStatus("RC");
                new MessageBuffer_dao(conexao).actualizarBuffer(buffer);
            }
        }
    } catch (SQLException ex) {
        logger.log(Level.ERROR, ex);
    }
}
```

```
    public Connection getConexao() throws SQLException {
        return conectionDB.getConnection();
    }

    private void processarStatusOfServerControl() {

        if (!control.getStatus().equalsIgnoreCase(EstadoAnterior)) {
            logger.log(Level.INFO, "Alteração do estado do servidor para ..." + control.getStatus().toUpperCase() + " ...");
            //noticar por email
            if (control.isNotificarEmail() && !EstadoAnterior.equalsIgnoreCase("RESTART")) {

                String mailMessage = "<!DOCTYPE html><html><body><h3>Alteração de estado do 'UMC SERVER',
Detalhes:</h3>"
                        + "<p>Esatdo Actual: ...#ESTADO#... <BR>"
                        + "Date Time: #DATE#<BR>"
                        + "User: #USER#<BR>"
                        + "</p><br>"
                        + "</body></html>";

                mailMessage = mailMessage.replaceAll("#ESTADO#", control.getStatus());
                mailMessage = mailMessage.replaceAll("#DATE#", control.getData());
                mailMessage = mailMessage.replaceAll("#USER#", control.getUser());

                MessageTemplate templEmail = new MessageTemplate_dao(conexao).systemTemplateEmail();

                if (templEmail != null) {
                    MessageBuffer buffer = new MessageBuffer();
                    buffer.setTemplate_id(templEmail.getId());
                    buffer.setBuilt_msg(mailMessage);
                    buffer.setDestination(control.getEmail());
                    buffer.setSubject("Notificação de Status UMCserver");
                    buffer.setSource_System("UMC SERVER");
                    buffer.setStatus("SC");
                    buffer.setOriginator("SERVER UMC");
                    buffer.setMsg_type("EMAIL");
                    new MessageBuffer_dao(conexao).notificar(buffer);
                } else {
                    new MessageTemplate_dao(conexao).createTemplEMAILsystem();
                }
            }
            //notificar por sms
            if (control.isNotificarSMS() && !EstadoAnterior.equalsIgnoreCase("RESTART")) {
                if (control.getTelemovel() != null) {
                    String smsMessage = "Alteracao de Status da UMCserver em #DATA# para #ESTADO#";
                    smsMessage = smsMessage.replace("#ESTADO#", control.getStatus().toUpperCase());
                    smsMessage = smsMessage.replace("#DATA#", control.getData());

                    MessageTemplate templSMS = new MessageTemplate_dao(conexao).systemTemplateSMS();

                    if (templSMS != null) {
                        MessageBuffer buffer = new MessageBuffer();
                        buffer.setTemplate_id(templSMS.getId());
                        buffer.setBuilt_msg(smsMessage);
                        buffer.setDestination(control.getTelemovel());
                        buffer.setSource_System("UMC SERVER");
                        buffer.setStatus("SC");
                        buffer.setOriginator("SERVER UMC");
                        buffer.setMsg_type("SMS");
                        buffer.setSubject("Notificacao de Status UMCserver");
```

```
                    new MessageBuffer_dao(conexao).notificar(buffer);
                } else {
                    new MessageTemplate_dao(conexao).createTemplSMSsystem();
                }
            }
        }
        EstadoAnterior = control.getStatus();
    }
    if (control.getStatus().equalsIgnoreCase("RESTART")) {
        new ServerControl_dao(conexao).setStatusRunning();
        restartThreads();
    }
}

@Override
public void run() {
    startThreads();
    while (status.getSisOverallStatus() != UMContactSistemStatusEnum.STOPPED) {
        try {

            monitor_Threads();
            arquivarBuffer();
            control = new ServerControl_dao(conexao).myControl();
            if (control.getStatus().equalsIgnoreCase("KILL")) {
                logger.log(Level.ERROR, "System killing by UMC " + control.getUser());
                System.exit(0);
            }
            processarStatusOfServerControl();
            Thread.sleep(1000);//lançar o monitor a cada 3 seg
            logger.log(Level.INFO, "Monitorando .....");

        } catch (InterruptedException ex) {
            logger.log(Level.ERROR, ex);
        }
    }
}
}
```

[FIM CLASS MONITOR]

[CLASS HttpSender]

```
package aplication.http;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Properties;
import org.apache.commons.codec.binary.Base64;
import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
import org.apache.http.client.HttpClient;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.DefaultHttpClient;
```

```
import org.apache.http.message.BasicNameValuePair;
import model.GatewaySMS;
import aplication.utils.GatewayResponse;
import model.MessageBuffer;
import org.json.simple.JSONObject;
import org.json.simple.JSONValue;

public class HttpSender {
    /*
     METODO PUBLICO PARA ENVIO DE SMS PARA DIFERENTES GATEWAYs.
     INVOCA O METODO CERTO PARA O GATEWAY ACTIVO NO MOMENTO, E CADA
     METODO É RESPONSAVEL POR IMPLEMENTAR A LOGICA DE ENVIO DO RESPECTIVO GATEWAY E
RETORNAR     O OBJECTO "GatewayResponse" COM OS ATRIBUTOS SETADOS
     */
    public GatewayResponse sendSMS(GatewaySMS gateway, MessageBuffer buffer) throws MalformedURLException,
IOException {

        if (gateway.getUrl_send().toLowerCase().contains("clickatell")) {
            return sendSMSclickatell(gateway, buffer.getUser_ID(), buffer.getId(), buffer.getDestination(), buffer.getBuilt_msg());
        } else if (gateway.getUrl_send().toLowerCase().contains("twilio")) {
            return sendSMStwilio(gateway, buffer);
        } else {
            GatewayResponse resposta = new GatewayResponse();
            resposta.setSendID(null);
            resposta.setUser_ID(buffer.getUser_ID());
            resposta.setBuffer_id(buffer.getGateway_id());
            resposta.setGateway_id(gateway.getId());
            resposta.setCod_erro(null);
            resposta.setStatus("FL");
            resposta.setMsg_erro("GATEWAY SMS NAO SUPORTADO PELO SISTEMA OU CONFIGURAÇÃO DE
GATEWAY INVALIDO.\n"
                    + "POR FAVOR CONTACTE O ADMINISTRADOR DO SISTEMA.");
            return resposta;
        }
    }

    private GatewayResponse sendSMSclickatell(GatewaySMS gway, int user_id, int MessageBufferID, String Contacto,
String Message) throws MalformedURLException, IOException {

        if (gway.getUser() != null && gway.getPassword() != null && gway.getHttp_api_id() != null && gway.getCallBack() !=
null
                && gway.getUrl_send() != null && MessageBufferID != 0) {
            GatewayResponse resposta = new GatewayResponse();
            //URL DE ENCAMINHAMENTO DA MENSAGEM
            String urlString = gway.getUrl_send();
            //preparcao da msg
            //subsituicao recomendado pelo proprio gateway
            Message = Message.replaceAll(" ", "+");
            //controlo de caracteres
            Message = Message.replaceAll("\n", "+");
            Message = Message.replaceAll("â", "a");
            Message = Message.replaceAll("Â", "A");
            Message = Message.replaceAll("Ã", "A");
            Message = Message.replaceAll("ã", "a");
            Message = Message.replaceAll("á", "a");
            Message = Message.replaceAll("à", "a");
            Message = Message.replaceAll("Á", "A");
            Message = Message.replaceAll("À", "A");
            Message = Message.replaceAll("ç", "c");
```

```
Message = Message.replaceAll("Ç", "C");
Message = Message.replaceAll("é", "e");
Message = Message.replaceAll("É", "E");
Message = Message.replaceAll("È", "E");
Message = Message.replaceAll("è", "e");
Message = Message.replaceAll("Ê", "E");
Message = Message.replaceAll("ê", "e");
Message = Message.replaceAll("º", "o");
Message = Message.replaceAll("ª", "a");
Message = Message.replaceAll("í", "i");
Message = Message.replaceAll("Í", "I");
Message = Message.replaceAll("ì", "i");
Message = Message.replaceAll("Ì", "I");
Message = Message.replaceAll("õ", "o");
Message = Message.replaceAll("Õ", "O");
Message = Message.replaceAll("ó", "o");
Message = Message.replaceAll("Ó", "O");
Message = Message.replaceAll("ò", "o");
Message = Message.replaceAll("Ò", "O");
Message = Message.replaceAll("ú", "u");
Message = Message.replaceAll("Ú", "Ú");
Message = Message.replaceAll("ù", "u");
Message = Message.replaceAll("Ù", "U");
Message = Message.replaceAll("û", "u");
Message = Message.replaceAll("Û", "U");
// os parametros a serem enviados

Properties parameters = new Properties();
parameters.setProperty("api_id", String.valueOf(gway.getHttp_api_id()));
parameters.setProperty("user", gway.getUser());
parameters.setProperty("password", gway.getPassword());
//indicativo sem sinal + a frent
if (Contacto.startsWith("00")) {
   parameters.setProperty("to", Contacto.replaceFirst("00", ""));
} else if (Contacto.startsWith("+")) {
   parameters.setProperty("to", Contacto.replace("+", ""));
} else {
   parameters.setProperty("to", Contacto);
}
parameters.setProperty("to", Contacto.replace("+", ""));
parameters.setProperty("from", gway.getFrom());
parameters.setProperty("text", Message);
parameters.setProperty("callback", gway.getCallBack());
if (Message.length() <= 160) {
   parameters.setProperty("concat", "1");
   resposta.setSegmento("1");
} else if (Message.length() > 160 && Message.length() < 306) {
   parameters.setProperty("concat", "2");
   resposta.setSegmento("2");
} else if (Message.length() > 306 && Message.length() < 459) {
   parameters.setProperty("concat", "3");
   resposta.setSegmento("3");
} else if (Message.length() > 459) {
   parameters.setProperty("concat", "N");
   if ((Message.length() % 153) != 0) {
      int seg = (Message.length()/153)+1;
      resposta.setSegmento(String.valueOf(seg));
   }else{
      int seg = (Message.length()/153);
```

```
            resposta.setSegmento(String.valueOf(seg));
        }
    }
    // o iterador, para criar a URL
    Iterator<Object> i = parameters.keySet().iterator();
    // o contador
    int counter = 0;

    // enquanto ainda existir parametros
    while (i.hasNext()) {

        // pega o nome
        String name = (String) i.next();
        // pega o valor
        String value = parameters.getProperty(name);

        // adiciona com um conector (? ou &)
        // o primeiro é ?, depois são &
        urlString += (++counter == 1 ? "?" : "&")
                + name
                + "="
                + value;
    }
    // cria o objeto url
    URL url = new URL(urlString);
    // cria o objeto httpurlconnection
    HttpURLConnection connection = (HttpURLConnection) url.openConnection();
    // conecta com a url destino
    connection.connect();
    // abre a conexão pra input
    BufferedReader input = new BufferedReader(new InputStreamReader(connection.getInputStream()));
    // le todo o conteudo de retorno
    StringBuilder inputString = new StringBuilder();
    String s = "";
    while (null != ((s = input.readLine()))) {
        inputString.append(s);
    }
    input.close();

    resposta.setGateway_id(gway.getId());
    resposta.setBuffer_id(MessageBufferID);
    resposta.setUser_ID(user_id);
    if (connection.getResponseCode() == 200 && connection.getResponseMessage().equals("OK")) {
        String conteudo = new String(inputString);
        if (conteudo != null) {
            if (conteudo.contains("ERR:")) {
                resposta.setStatus("FL");
                resposta.setCod_erro(conteudo.substring(5, 8));
                resposta.setMsg_erro(conteudo);
                resposta.setSendID(null);
            } else if (conteudo.contains("ID:")) {
                resposta.setStatus("GW");
                resposta.setCod_erro(null);
                resposta.setMsg_erro(null);
                resposta.setSendID(conteudo.substring(4, conteudo.length()));
                resposta.setSegmento("1");
            }
        }
    } else {
```

```
            resposta.setStatus("SC");
            resposta.setCod_erro(null);
            resposta.setMsg_erro(null);
            resposta.setSendID(null);
        }

        return resposta;
    }
    return null;
}

private GatewayResponse sendSMStwilio(GatewaySMS gway, MessageBuffer buffer) throws IOException {

    GatewayResponse resposta = new GatewayResponse();
    resposta.setGateway_id(gway.getId());
    resposta.setBuffer_id(buffer.getId());
    resposta.setStatus("GW");
    resposta.setUser_ID(buffer.getUser_ID());
    resposta.setTemplate_id(buffer.getTemplate_id());

    if (gway.getAccountSid() != null && gway.getAuthoToken() != null && gway.getFrom() != null
            && gway.getCallBack() != null && gway.getUrl_send() != null && gway.getApplicationSId() != null) {

        @SuppressWarnings("deprecation")
        HttpClient client = new DefaultHttpClient();

        String URL = gway.getUrl_send();
        /*verifica se a url esta de acordo com as normas necessariais
         para o bom funcionamento do sistema*/

        if (!URL.endsWith("/Accounts/{AccountSid}/Messages.json")) {
            if (URL.endsWith("/Messages")) {
                URL += ".json";
            } else if (URL.endsWith(".xml")) {
                URL = URL.replace(".xml", ".json");
            }
        }
        /*garantir de que a url tem o formato certo, ou seja o formato ideal para o bom
         funcionameno do sistema*/
        if (URL.contains("/Accounts/{AccountSid}/Messages.json")) {
            if (URL.contains("{AccountSid}") && gway.getAccountSid() != null) {
                URL = URL.replace("{AccountSid}", gway.getAccountSid());
            }
            HttpPost post = new HttpPost(URL);
            /* Autenticando no serviço */
            post.setHeader("Authorization", "Basic " + myAuthentication(gway.getAccountSid(), gway.getAuthoToken()));
            /* Configura os parâmetros do POST */
            List<NameValuePair> urlParameters = new ArrayList<>();
            //controlo de caracteres por questao de preço dos SMS

            /*QUALQUER MSG SMS COM MAIS DE 70 CARACARES, E COM CARACTERES NAO GSM,
             SAO SAO COBRADOS O DOBRO OU MAIS VEZES O PREÇO NORMAL DE UM SMS. POIS SAO
NECESSARIOS VARIOS
             CARACTERES PARA IDENTIFICAR UMA CEDILHA OU ACENTO UTILIZADO.
             CARACTERES NAO GSM SAO: CEDILHAS, ACENTOS, etc*/
            String Message = buffer.getBuilt_msg();
            if (Message.length() > 69) {
                if (buffer.isControl_Caracteres()) {
                    Message = Message.replaceAll("\n", " ");
```

```
            Message = Message.replaceAll("â", "a");
            Message = Message.replaceAll("Â", "A");
            Message = Message.replaceAll("Ã", "A");
            Message = Message.replaceAll("ã", "a");
            Message = Message.replaceAll("á", "a");
            Message = Message.replaceAll("à", "a");
            Message = Message.replaceAll("Á", "A");
            Message = Message.replaceAll("À", "A");
            Message = Message.replaceAll("ç", "c");
            Message = Message.replaceAll("Ç", "C");
            Message = Message.replaceAll("é", "e");
            Message = Message.replaceAll("É", "E");
            Message = Message.replaceAll("È", "E");
            Message = Message.replaceAll("è", "e");
            Message = Message.replaceAll("Ê", "E");
            Message = Message.replaceAll("ê", "e");
            Message = Message.replaceAll("º", "o");
            Message = Message.replaceAll("ª", "a");
            Message = Message.replaceAll("í", "i");
            Message = Message.replaceAll("Í", "I");
            Message = Message.replaceAll("ì", "i");
            Message = Message.replaceAll("Ì", "I");
            Message = Message.replaceAll("õ", "o");
            Message = Message.replaceAll("Õ", "O");
            Message = Message.replaceAll("ó", "o");
            Message = Message.replaceAll("Ó", "O");
            Message = Message.replaceAll("ò", "o");
            Message = Message.replaceAll("Ò", "O");
            Message = Message.replaceAll("ú", "u");
            Message = Message.replaceAll("Ú", "Ú");
            Message = Message.replaceAll("ù", "u");
            Message = Message.replaceAll("Ù", "U");
            Message = Message.replaceAll("û", "u");
            Message = Message.replaceAll("Û", "U");
        }
    }

    urlParameters.add(new BasicNameValuePair("Body", Message));
    if (buffer.getDestination().startsWith("00")) {
        buffer.setDestination(buffer.getDestination().replaceFirst("00", "+"));
        urlParameters.add(new BasicNameValuePair("To", buffer.getDestination()));
    } else if (buffer.getDestination().startsWith("+")) {
        urlParameters.add(new BasicNameValuePair("To", buffer.getDestination()));
    } else {
        urlParameters.add(new BasicNameValuePair("To", "+" + buffer.getDestination()));
    }
    urlParameters.add(new BasicNameValuePair("From", gway.getFrom()));
    urlParameters.add(new BasicNameValuePair("StatusCallback", gway.getCallBack()));
    urlParameters.add(new BasicNameValuePair("ApplicationSid", gway.getApplicationSId()));
    post.setEntity(new UrlEncodedFormEntity(urlParameters, "UTF-8"));
    /* Efetua o POST */
    HttpResponse conexao = client.execute(post);

    if (conexao.getStatusLine().getReasonPhrase().equals("OK") ||
conexao.getStatusLine().getReasonPhrase().equals("CREATED")) {
        BufferedReader input = new BufferedReader(new InputStreamReader(conexao.getEntity().getContent()));
        StringBuilder inputString = new StringBuilder();
        String line = "";
        while ((line = input.readLine()) != null) {
```

```
                inputString.append(line);
            }
            input.close();
            if (new String(inputString) != null) {
                JSONObject obj = (JSONObject) JSONValue.parse(new String(inputString));
                if (obj.get("sid") != null) {
                    resposta.setSendID(obj.get("sid").toString());
                    resposta.setMsg_erro(null);
                    resposta.setCod_erro(null);
                }
                if (obj.get("num_segments") != null) {
                    resposta.setSegmento(obj.get("num_segments").toString());
                }
            }
          } else {
            resposta.setSendID(null);
            resposta.setStatus("FL");
            resposta.setMsg_erro(conexao.getStatusLine().getReasonPhrase());
            resposta.setCod_erro(String.valueOf(conexao.getStatusLine().getStatusCode()));
          }
        } else {
          resposta.setSendID(null);
          resposta.setStatus("FL");
          resposta.setMsg_erro("URL_SEND nao suportado pelo sistema! Alterar para
\".../Accounts/{AccountSid}/Messages.json\"");
          resposta.setCod_erro(null);
        }
        return resposta;
      } else {
        resposta.setMsg_erro("CONFIGURÇÃO DE GATEWAY INVALIDO");
        resposta.setSendID(null);
        resposta.setStatus("FL");
        resposta.setCod_erro(null);
        return resposta;
      }
    }

    private String myAuthentication(String ACCOUNTSID, String AUTHTOKEN) {
      return new Base64().encodeToString((ACCOUNTSID + ":" + AUTHTOKEN).getBytes());
    }
}
```

[FIM CLASS HttpSender]

[CLASS SmtpSender][

```
 //CLASS RESPONSAVEL PELO ENVIO DOS DIVERSOS EMAILS DO SISTEMA.
package aplication.smtp;

import org.apache.commons.mail.EmailException;
import org.apache.commons.mail.HtmlEmail;
import model.ConfiguracaoSMTP;

public class SmtpSender {

    public void enviarEmail(ConfiguracaoSMTP configSMTP, String para, String assunto, String menssagem) throws
EmailException{

        HtmlEmail EMAIL = new HtmlEmail();
        EMAIL.setDebug(true);
```

```
        EMAIL.setHostName(configSMTP.getSERVIDOR_OUT());
        EMAIL.setAuthentication(configSMTP.getEMAIL(), configSMTP.getPASSWORD());
        EMAIL.setSSL(configSMTP.isSSL());
        EMAIL.setSmtpPort(configSMTP.getPORTA());
        EMAIL.addTo(para);
        EMAIL.setFrom(configSMTP.getEMAIL(), configSMTP.getFROM());
        EMAIL.setSubject(assunto);
        EMAIL.setContent(menssagem, "text/html");
        EMAIL.send();
    }
}
```

FIM DA CLASS SmtpSender]

II. Manual do utilizador UMCont@ct

MANUAL
DO
UTILIZADOR

UMCont@ct

O mundo das alertas ao seu alcance!

UMCont@ct é um projeto desenvolvido a pensa em si que pretende atingir o seu publico alvo com diversas alerta ou mensagens de diversos assuntos, de forma económico, eficiente e sobretudo o mais rápido possível através de mensagens SMS.

Projetado para uma rápida e simples integração com a sua empresa ou projetos, a UMC fornece meios que permite que as suas aplicações desenvolvidas em diversas linguagens de programação desfrute das vantagens das mensagens SMS como meio de comunicação com os diversos *stakeholders* da sua empresa.

Índice do Manual

1. AVISO IMPORTANTE 2

2. REGISTRO NO SISTEMA 2

2.1. RECUPERAÇÃO DE PASSWORD 3

3. CONFIGURAÇÃO DE LIMITES DE UTILIZAÇÃO 3

4. CRIAÇÃO DE TEMPLATES 5

5. IMPORTAÇÃO DE CONTACTOS 6

6. ENVIO DE MENSAGENS 7

6.1. REQUISITOS NECESSÁRIOS 7

6.2. ENVIO PELA CONTA DO CLIENTE 8

6.2.1. Envio para contactos importados 9

6.3. AGENDAMENTO DE MENSAGENS 9

6.4. ENVIO DA MENSAGENS POR APLICAÇÕES EXTERNAS 11

6.4.1. Web Service 11

6.4.2. POST e GET HTTP 11

7. RESPOSTAS DE SOLICITAÇÕES DE ENVIO 12

1. Aviso importante

Ao longo deste manual de utilizador será utilizado o URL utilizado na fase de teste do sistema (http://sigaum.servehttp.com/um.contact.admin/index.php), porem na fase de produção a URL poderá sofrer alterações, pelo que pedimos a compreensão de todos.

2. Registro no sistema

O registo no sistema procede como qualquer outro sistema. Ao utilizador é requisitado um conjunto de informações relevantes a sua identificação no sistema, sendo os mais importantes o username, o password e o email do utilizador.

Na figura 1 é apresentado um exemplo registo de um cliente no sistema:

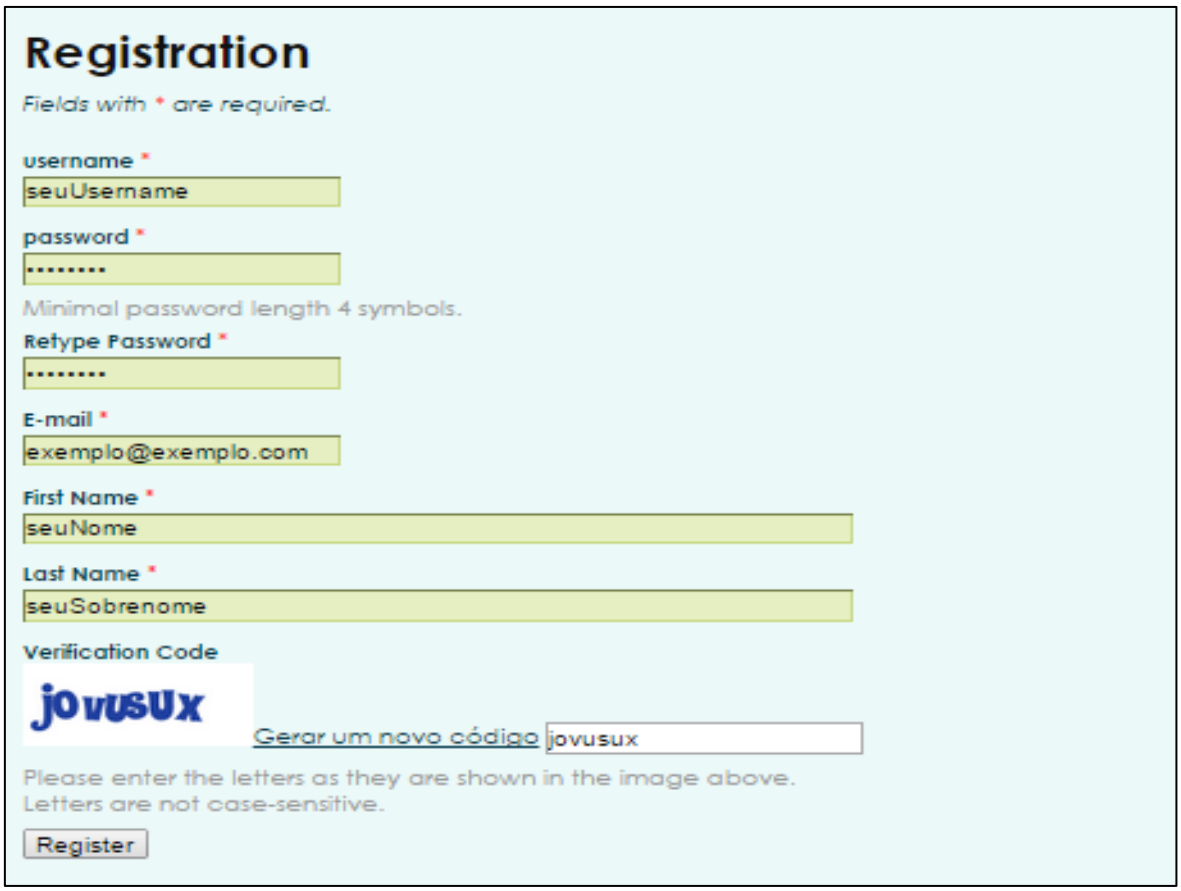

Figura 31- Registo no sistema

2.1. Recuperação de password

Ao esquecer o password o sistema disponibiliza aos clientes um mecanismo de recuperação de password através do seu **email** ou **username** em uso no sistema.

Para recuperar o password o utilizador deve clicar em *Lost Password?* na página de login do sistema, informado o seu email utilizado no sistema, será envia um link que permite que o utilizador redefine o seu password.

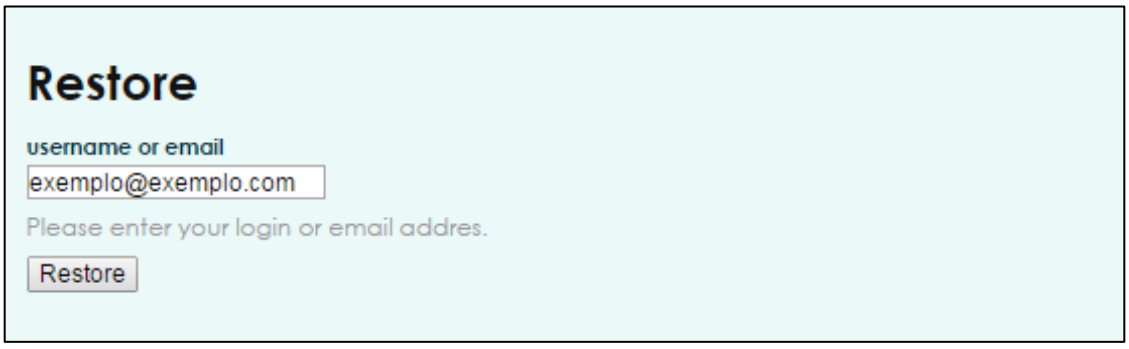

Figura 32 - Recuperação de Password

3. Configuração de limites de utilização

A pensar num maior controlo de crédito de cada cliente, a UMCont@ct permite que cada utilizador configure os limites de utilização por dia e por hora do crédito de serviço SMS e Email. No momento de cadastrar no sistema, é criado de forma automático os limites de utilização do cliente, por padrão zero (0). O campo **User Key** representa a chave (userKey) que o utilizador utilizará em caso de conexão do seu sistema com a UMCont@ct.

As configurações dependem de cliente para cliente e do volume de mensagens que pretende ser enviadas por dia e hora através da sua conta.

A atribuição de crédito ao cliente é de competência do utilizador administrador, pelo que o campo MAX SMS e MAX EMAIL não podem ser alterados pelos clientes.

Na figura 3 é apresentado um exemplo de configuração de limites:

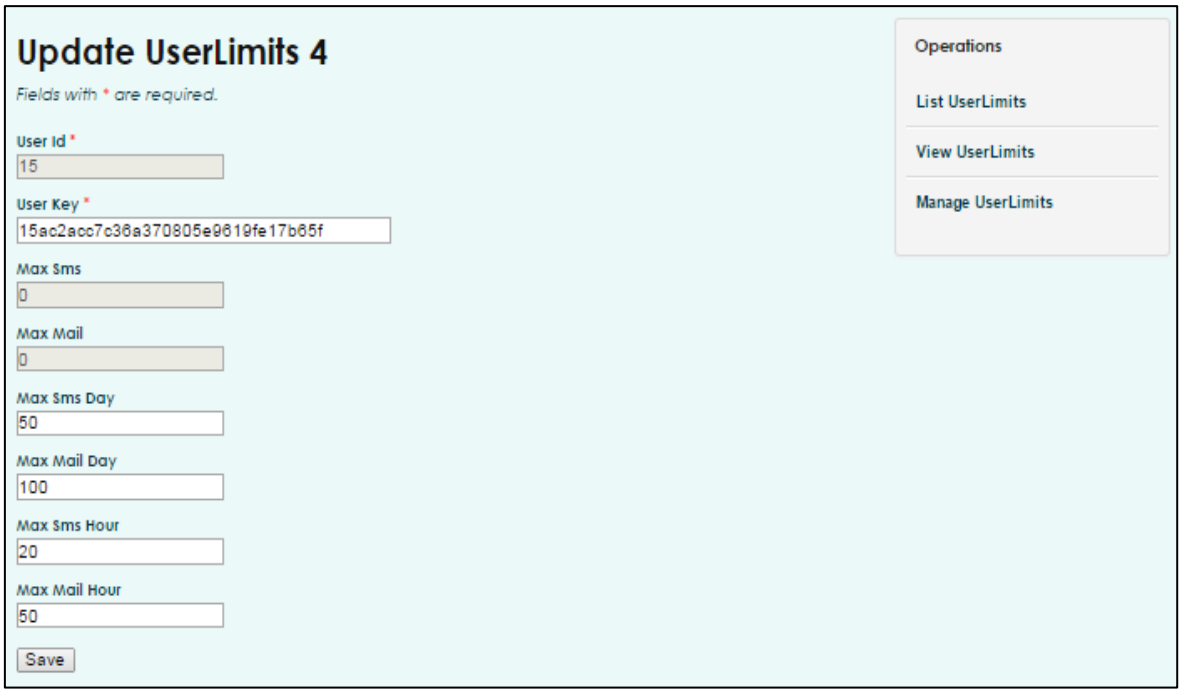

Figura 33 - Configuração dos limites de envio de alertas

Descrição dos campos:

User Key – chave utilizado pelo utilizador para fazer pedidos externos de mensagens;

Max sms – quantidade de crédito SMS disponível na conta do utilizador;

Max Mail – quantidade de email disponível na conta do utilizador;

Max sms Day – quantidade de SMS permitido para envio diário;

Max Mail Day – quantidade máxima de envio de email por dia;

Max sms Hour – quantidade de envio de SMS permitido na conta;

Max Mail Hour – quantidade de envio de email permitido;

4. Criação de Templates

O sistema funciona a base de Templates de mensagens. Os templates contem diversas informações a destacar o tipo de mensagem (SMS ou Email), o tipo de envio (instantâneo ou agendado), o utilizador do Templates, a utilização ou não de controlo de caracteres em SMS, o sistema de origem da mensagem, o dia e a hora de envio das mensagens, entre outros. Parte dessas informações são passadas para cada buffer de mensagem que utiliza esse template.

Ao ser criado um novo utilizador no sistema, um template de SMS instantâneo é criado e atribuído a esse utilizador. Podendo este criar novos templates de acordo com a sua necessidade.

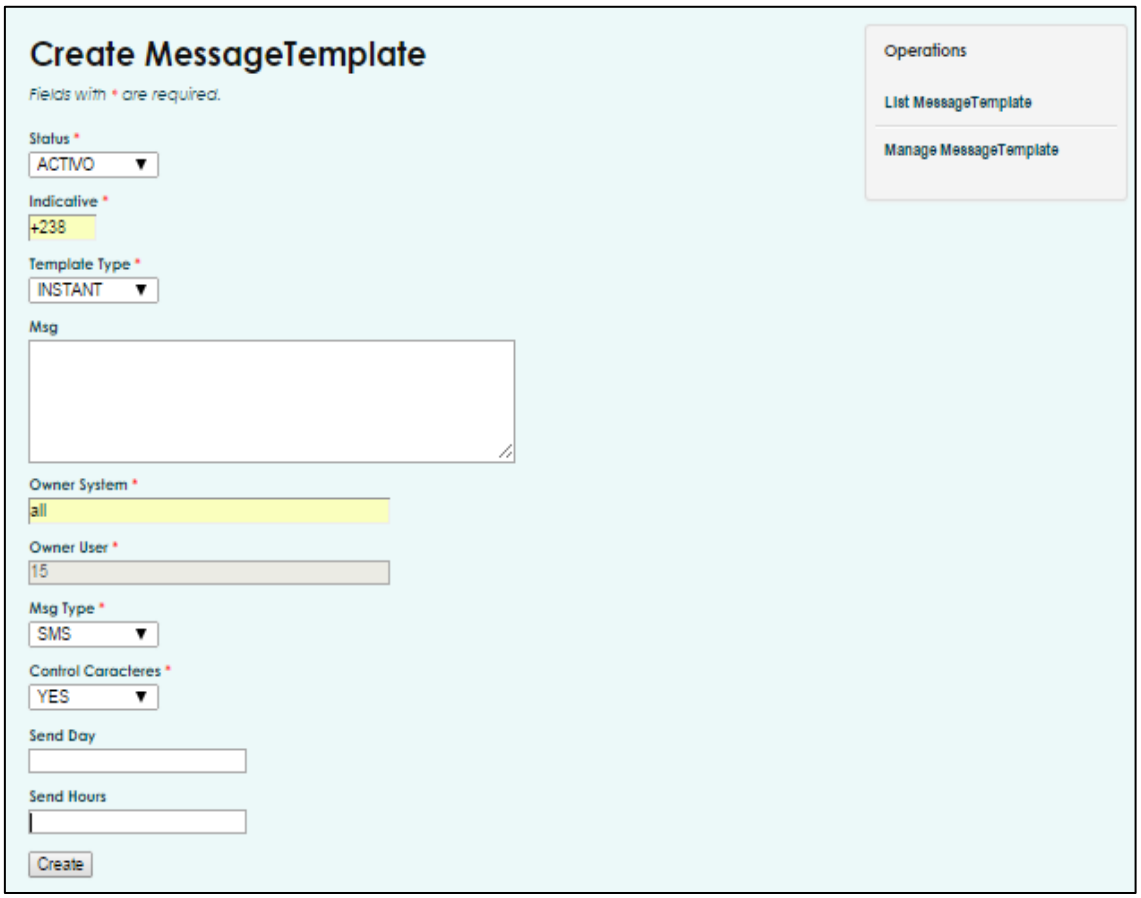

Figura 34 - Criação de template

5. Importação de contactos

Para processar envio de mensagens a um grande volume de contatos, o sistema fornece meios dos utilizadores importar contatos telemóveis de ficheiros Excel.

Na importação é requisitado uma chave que identifica os contactos no sistema, chave que será sempre pedido a quando de envio de mensagens a contatos importados.

Para importar contactos prima o botão "***Escolher ficheiro",*** escolha o ficheiro a importar, digite a chave de identificação dos contactos e click em "***Importar".***

Figura 35 - Importação de contactos

O nome é irrelevante na importação, sendo vazio em caso de sua inexistência.

6. Envio de mensagens

No envio de mensagens o utilizador deve ter em atenção a quantidade de crédito disponível, os limites de envio por hora e dia e o template utilizado, sendo o assunto relevante apenas para mensagens de email.

Para mensagens SMS o utilizador deve ter em consideração de que ao utilizar caracteres especiais no corpo da mensagem, poderá ser cobrado o valor superior a um crédito, dependendo do número de caracteres especiais.

O número de caracteres suportados (sem caracteres especiais) é de 160, sendo com caracteres especiais o máximo de caracteres é de 70 por segmento de SMS.

Quando utiliza sistema externos para interligação com a UMContact, sempre que efetuar uma requisição de envio de mensagens, recebe um feedback por parte do sistema. Para mais informações sobre as respostas do serviço veja o tópico ***Respostas de solicitações de envio***.

6.1. Requisitos necessários

Para que seja possível enviar mensagens pela UMCont@ct, um conjunto de requisitos devem sem reunidas:

1º Ser utilizador do sistema;

2º Ter crédito disponível na UMC;

3º Configurar os limites de envio;

4º Ter template disponível;

6.2. Envio pela conta do cliente

Após as devidas configurações da conta, ao utilizador é disponibilizado um formulário de envio de mensagem individual e outro para envio de mensagem de grupo, aos contactos importados.

Para mensagem individual Selecione o menu ***Message*** > ***MSG Buffer*** > ***Create Message Buffer*** e preenche as informações necessárias e clica em ***Create.***

Na figura 6 é apresentado um exemplo de criação de SMS individual:

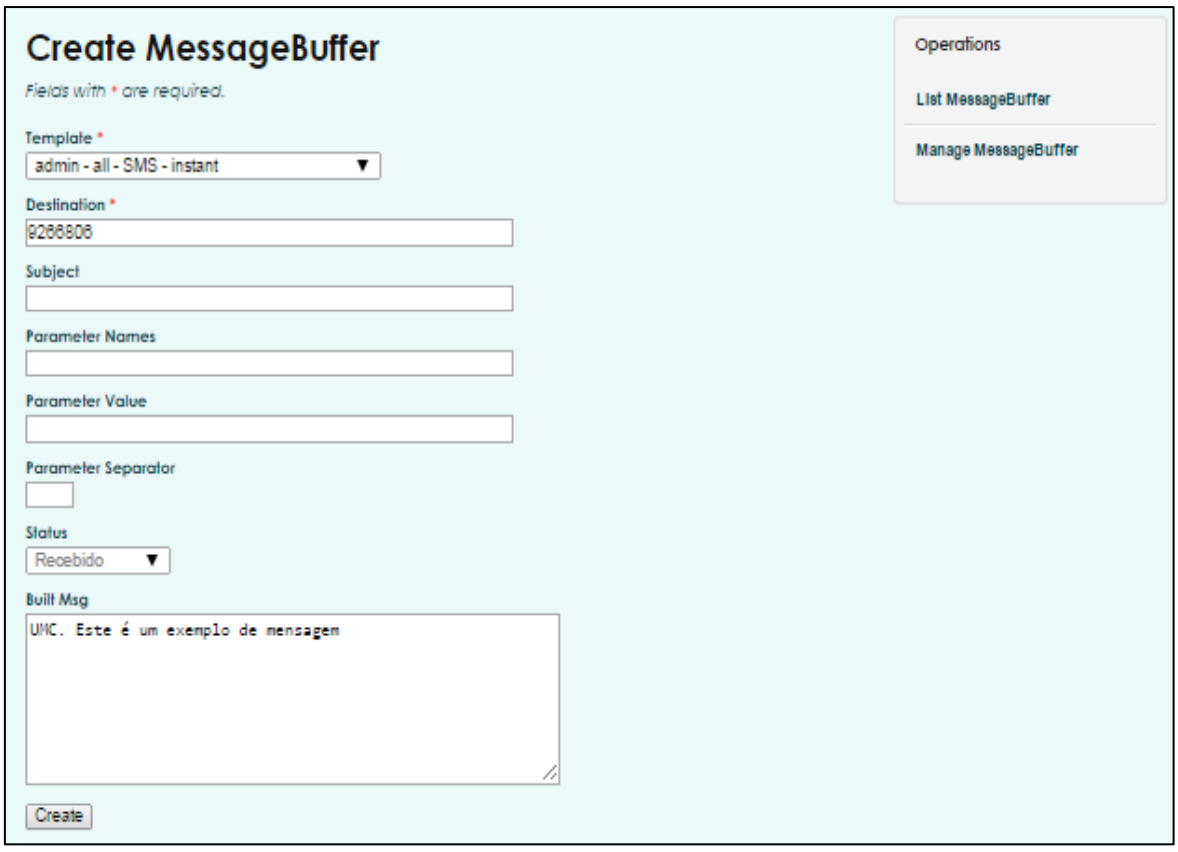

Figura 36 - criação de mensagem SMS individual

6.2.1. Envio para contactos importados

Mensagem de grupo está disponível apenas para SMS e aos contactos importados. Para envio de mensagens a contactos importados do Excel vá ao menu ***Message*** > ***MSG Buffer*** > ***Create msg to imported contacts*** e preenche as informações necessárias.

- **Template** – template do utilizador a utilizado no processo de envio
- **Chave** – chave que identifica os contactos no sistema, cada chave é um grupo de contactos.
- **Built Msg** – mensagem a ser enviada ao grupo.

A figura 7 demostra a criação de mensagens de grupo a contactos previamente importados na chave ***IG_04***:

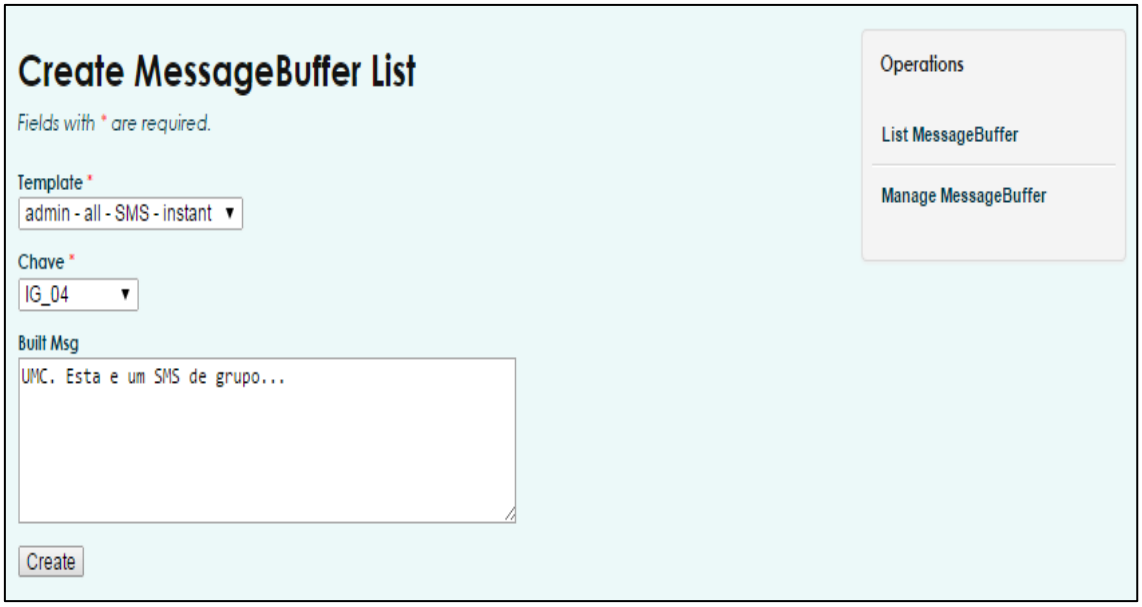

Figura 37 - criação de mensagem de grupo

6.3. Agendamento de mensagens

O processo de agendamento de mensagens dá pela utilização ou não de templates de mensagens com essa funcionalidade. Assim para agendar mensagens o utilizador deve

primeiramente configurar um template do tipo *normal*, fornecendo o dia e a hora de envio de mensagens originada a partir do template.

O primeiro dia da semana domingo é representado por 1, sendo segunda 2, terça 3 e assim sucessivamente. A hora deve ser a hora exata do envio, exemplo 1 para 01:00:00, 2 para 02:00:00 e assim por diante. Para intervalo de tempo deve utilizar o (-) para separa os números do intervalo, exemplo 2-4 para segunda a quarta para dia, e 17-20 para 17:00:00 as 20:00:00 para hora.

A seguir figura 8 é apresentado um exemplo de template de agendamento de mensagens para o intervalo de segunda a quinta-feira das 16 as 18 horas:

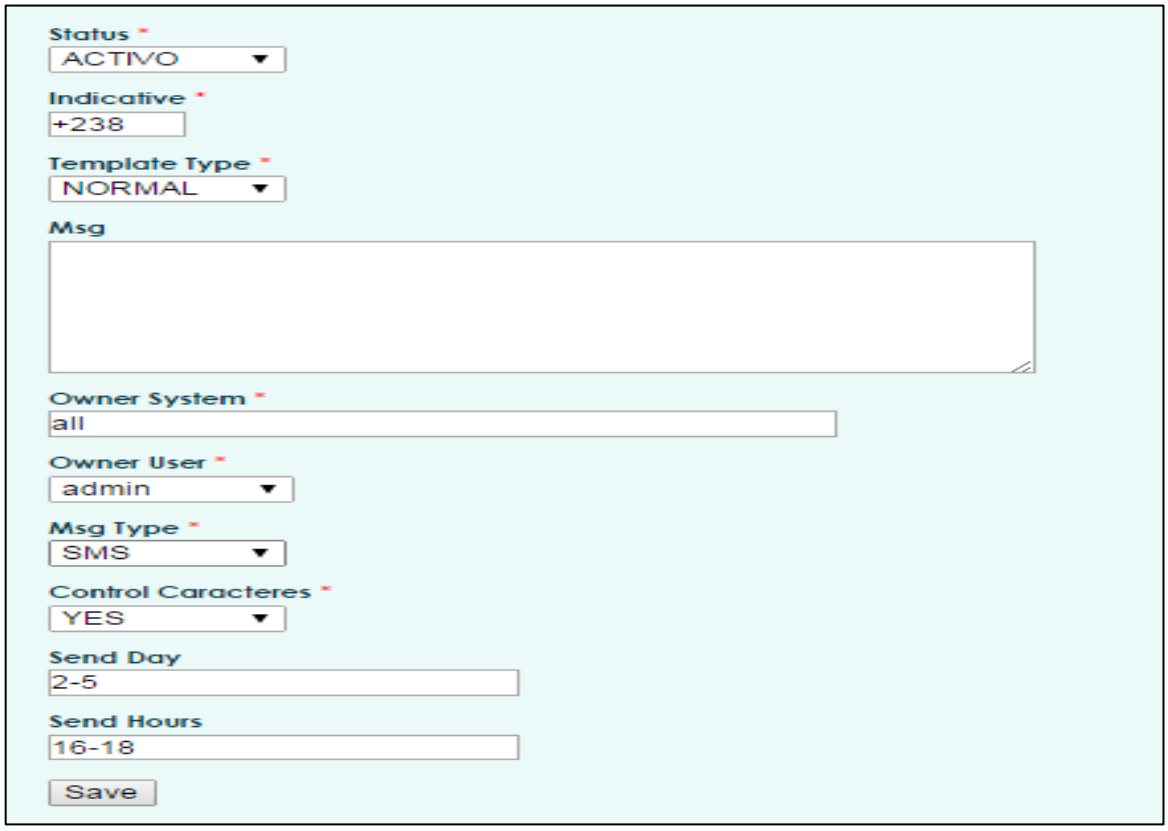

Figura 38 - configuração de template de agendamento

6.4. Envio da mensagens por aplicações externas

O foco principal da UMCont@ct é a de enviar alerta diversas de outras aplicações externas, funcionando como um gateway de alertas. Para isso o sistema fornece um Web Service e POST e GET http para uma rápida e segura comunicação entre a sua aplicação e a UMC, desfrutando das diversas vantagens dos nossos serviços de alerta.

6.4.1. Web Service

Para aceder os recursos do nosso sistema os utilizadores do mesmo poderão utilizar o Web Service http://URL_do_site/index.php/service/quote/wsdl:

Ao aceder ao link do Web service o cliente terá acesso ao WSDL (Web Service Description Language) que fornece informações da utilização do Web Service.

Aos utilizadores é fornecido métodos para enviar SMS e Email tanto instantâneo como agendado, consultar crédito no sistema, ver status de uma mensagem no sistema.

6.4.2. POST e GET HTTP

Se preferir o utilizador poderá utilizar o protocolo http como meio de comunicação com a UMC, para isso foi criado e disponibilizado aos clientes um conjunto de métodos que podem ser acedidos tanto por POST como por GET.

A seguir é apresentado um conjunto de GET http que podem ser utilizados também em POST:

Ex1: Envio de mensagens SMS instantâneas:

http://url_do_site_do_sistema/index.php/messageBuffer/ExtAppMsgInstant?User=xxxxx&UserKey=xxxxxx&Template=xx&Destination=xxxxxxx&Msg=xxxxxxxxx

Ex2: Envio de mensagens SMS agendados:

http://URL_do_site_do_sistema/index.php/messageBuffer/ExtAppMsgNormal?User=xxxxppMsgNormal?User=xxxxx&UserKey=xxxxxx&Template=xx&Destination=99xxx&Key_names=xxxxxxxx&Key_separator=x&Key_values=xxxxxxx

Ex3: Envio de mensagens Email Instantâneo:

http://URL_do_site_do_sistema/index.php/messageBuffer/ExtAppMsgInstant?User=xxxxx&UserKey=xxxxxx&Template=xx&Destination=xxxxx@xxxx&Msg=xxxxxxxxxxxx&Subject=xxxxxxxx

Ex3: Envio de mensagens Email Instantâneo:

http://URL_do_site_do_sistema/index.php/messageBuffer/ExtAppMsgNormal?User=xxxxx&UserKey=xxxxxx&Template=xx&Destination=xxxxx@xxxxx&Subject=xxxxxxKey_names=xxxxxxxx&Key_separator=x&Key_values=xxxxxxx

Os espaços existentes no corpo da mensagem podem ser substituídos pelo sinal mais (+), no caso de utilização do método GET. Pelos exemplos acima poderá ver que o assunto da mensagens (*Subject*) é relevante apenas para mensagens de email.

7. Respostas de solicitações de envio

Todos os pedidos GET e POST endereçados ao sistema tem como resposta um JSON contendo as descrições do resultado do processamento do pedido no sistema.

A seguir é apresentado um exemplo de resposta de um pedido de mensagem feito com sucesso e outro com erro devido a um parâmetro vazio ou por falta de um parâmetro:

Ex1: Requisição feito com sucesso:

{"***status***":"OK", "***code***":"000", "***message***":"Mensagem Recebida ", "***msgId***":"uPlecd3Td6ht5sbY#kp"}

Ex2: Requisição feito com insucesso:

{"***status***":"erro","***code***":"001","***message***":"Parametro vazio","***msgId***":null}

Os mesmos parâmetros são retornados quando o utilizador utiliza Web service como meio de comunicação entre os sistemas. Neste caso as informações são passadas através do protocolo padrão de comunicação de Web service, o SOAP baseado em XML.

Tendo em vista a facilitação de programadores que utiliza o nosso sistema, um conjunto de códigos foram utilizados para ajudar na tomada de decisão sobre diversos casos que pode acontecer a quando de uma requisição no sistema. No quadro a seguir é apresentado a descrição dos códigos:

CÓDIGO	MENSAGEM	DESCRIÇÃO
"000"	"Mensagem Recebida" "Creditos do cliente" "Status atual"	Código de sucesso resultante da aceitação do pedido do cliente no sistema. Retomado sempre que há sucesso no pedido do cliente.
"001"	"Parametro vazio"	Código de erro que resulta quando o pedido é feito com um parâmetro vazio ou por falta desta.
"002"	"Autenticacao invalida"	Código de erro resultante quando a autenticação do utilizador no sistema não é bem-sucedido. *username* e o *userKey* inválidos.
"003"	"Template invalido"	Resulta quando o template não existe no sistema, ou se existe não pertence ao utilizador que fez a requisição, ou se pertence, não é do tipo de mensagem que se pretende enviar
"004"	"Erro interno. Por favor tente novamente"	Código retomado quando o motivo é algum problema interno ao sistema, ou o erro é desconhecido
"005"	"MsgId invalido para UMC"	Código de erro quando a msgId que se utiliza na requisição de status de mensagem não está dentro do padrão se código de mensagens retomado
"006"	"MsgId desconhecido"	Código de erro originado pela não existência da msgId no sistema

Tabela 8 - Códigos de requisição